การประยุกต์ใช้ปัญญาประดิษฐ์สำหรับปัญหาการจัดตารางงาน ที่มีหลายวัตถุประสงค์ในการประมวลผลแบบกลุ่มเมฆ

APPLIED ARTIFICIAL INTELLIGENCE FOR MULTI-OBJECTIVE TASK SCHEDULING PROBLEM IN CLOUD COMPUTING

บุญหทัย เครือแก้ว

BOONHATAI KRUEKAEW

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
ปริญญาดุษฎีบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์
ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2565

KMITL-2022-SC-D-002-062

APPLIED ARTIFICIAL INTELLIGENCE FOR MULTI-OBJECTIVE TASK SCHEDULING PROBLEM IN CLOUD COMPUTING

BOONHATAI KRUEKAEW

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE SCHOOL OF SCIENCE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2022

KMITL-2022-SC-D-002-062

หัวข้อวิทยานิพนธ์	การประยุกต์ใช้ปัญญาประดิษฐ์สำหรับปัญหาการจัดตารางงานที่มีหลายวัตถุประสงค์ในการประมวลผลแบบกลุ่มเมฆ
ชื่อนักศึกษา	นางสาวบุญหทัย เครือแก้ว
รหัสประจำตัว	57605018
ปริญญา	ปรัชญาดุษฎีบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชา	วิทยาการคอมพิวเตอร์
พ.ศ.	2565
อาจารย์ที่ปรึกษาวิทยานิพนธ์	ผศ.ดร.วรางคณา กิ้มปาน

บทคัดย่อ

การจัดสรรภาระงานในการประมวลผลแบบกลุ่มเมฆเป็นปัญหาที่ท้าทาย โดยเฉพาะอย่างยิ่งในบริการด้านโครงสร้างฮาร์ดแวร์พื้นฐาน (Infrastructure as a Service) ปัญหาที่ไม่ควรเกิดขึ้นในการเข้าใช้งานการประมวลผลแบบกลุ่มเมฆ คือการที่โฮสต์หรือเครื่องแม่ข่ายตัวหนึ่งตัวใดทำงานหนักเกินไปหรือน้อยเกินไป ซึ่งผลต่อเวลาในการทำงานหรืออาจส่งผลให้ระบบขัดข้อง เพื่อป้องกันไม่ให้เกิดปัญหาดังกล่าวจึงควรการจัดตารางเวลาการเข้าใช้งานให้เหมาะสมกับสภาพแวดล้อม เพื่อให้มีการกระจายงานไปยังทรัพยากรแบบสมดุล ซึ่งการทำให้ระบบสมดุลนั้นควรตรวจสอบให้แน่ใจว่ามีการใช้เครื่องคอมพิวเตอร์เสมือน (Virtual Machines) ทั้งหมดอย่างเหมาะสม ในงานวิจัยนี้ได้เสนอวิธีการจัดตารางงานที่เป็นอิสระต่อกันในประมวลผลแบบกลุ่มเมฆ โดยมุ่งเน้นที่วิธีการจัดตารางงานแบบที่มีหลายวัตถุประสงค์ อีกทั้งยังได้มีการนำวิธีการทางปัญญาประดิษฐ์เข้ามาช่วยในการแก้ไขปัญหา โดยเรียกวิธีที่นำเสนอนี้ว่า “MOABCQ” ซึ่งเป็นการประยุกต์ใช้ขั้นตอนอาณานิคมผึ้งเทียม (Artificial Bee Colony algorithm) ควบคู่กับการใช้ขั้นตอนวิธีการเรียนรู้แบบคิว (Q-learning algorithm) วิธีการที่เสนอมีวัตถุประสงค์เพื่อเพิ่มประสิทธิภาพในการจัดตารางงานและการใช้ทรัพยากร และสร้างสมดุลระหว่างเครื่องคอมพิวเตอร์เสมือน โดยพิจารณาจากเวลารวมในการทำงาน ค่าใช้จ่ายในการใช้งาน การใช้ทรัพยากร เป็นข้อจำกัดของการพิจารณาพร้อมกัน การวิเคราะห์ประสิทธิภาพของวิธีการที่นำเสนอถูกเปรียบเทียบโดยใช้ CloudSim และวิธีการจัดตารางงานอื่นคือ วิธี Max-Min วิธี FCFS วิธี HABC_LJF วิธี Q-learning วิธี MOPSO และวิธี MOCS ผลการทดลองพบว่าการจัดตารางงานโดยใช้วิธี MOABCQ มีประสิทธิภาพที่ดีกว่าวิธีการอื่นๆ ในแง่ของช่วยลดเวลาในการทำงาน ลดค่าใช้จ่าย ลดค่าความไม่สมดุลของการกระจาย และช่วยเพิ่มประสิทธิภาพการทำงานในระบบได้เป็นอย่างดี

คำสำคัญ: การกระจายงานแบบสมดุล การประมวลผลแบบกลุ่มเมฆ การเรียนรู้แบบเสริมกำลัง ขั้นตอนอาณานิคมผึ้งเทียม ปัญญาประดิษฐ์

Thesis Title	Applied Artificial Intelligence for Multi-Objective Task Scheduling Problem in Cloud Computing
Student Name	Miss Boonhatai Kruekaew
Student ID	57605018
Degree	Doctor of Philosophy (Computer Science)
Department	Computer Science
Year	2022
Thesis Advisor	Asst.Prof.Dr.Warangkhana Kimpan

Abstract

Workload balancing in cloud computing is still challenging problem, especially in Infrastructure as a Service (IaaS) in the cloud model. A problem that should not occur during cloud access is a host or server being overloaded or underloaded, which may affect the processing time or may result in a system crash. Therefore, to prevent these problems, an appropriate schedule of access should be considered so that the system can distribute tasks across all available resources, which is called load balancing. The load balancing technique should ensure that all Virtual Machines (VMs) are used appropriately. In this research, an independent task scheduling approach in cloud computing environment is proposed using a Multi-objective task scheduling optimizationm model based the Artificial Bee Colony algorithm (ABC) adapted with the implementation of the Q-learning algorithm, which are an artificial Intelligence method, called the MOABCQ method. The proposed method aims to optimize scheduling and resource utilization, maximize VM throughput, and create load balancing between VMs based on makespan, cost, and resource utilization, which are limitations of concurrent considerations. Performance analysis of the proposed method was compared using CloudSim with the existing load balancing and scheduling algorithms: Max-Min, FCFS, HABC_LJF, Q-learning, MOPSO, and MOCS algorithms. The experimental results indicated that the algorithms used MOABCQ approach outperformed the other algorithms in terms of reducing makespan, reducing cost, reducing degree of imbalance, increasing throughput and average resource utilization.

Keywords : Load Balance, Cloud Computing, Reinforcement Learning, Artificial Bee Colony (ABC) Algorithm, Artificial Intelligence

กิตติกรรมประกาศ

วิทยานิพนธ์นี้ ตั้งแต่เริ่มต้นจนสำเร็จลุล่วงนั้น นับว่าเป็นความภาคภูมิใจของผู้วิจัยเป็นอย่างมาก หากลำพังผู้วิจัยเพียงผู้เดียวคงไม่สามารถดำเนินงานจนประสบความสำเร็จเช่นนี้ได้ ทั้งนี้เพราะได้รับความกรุณาจากบุคคลหลายๆ ท่านดังนี้

ขอขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร.วรางคณา กิ้มปาน อาจารย์ที่ปรึกษา ที่ได้สละเวลาให้คำแนะนำ ดูแลเอาใจใส่ รวมทั้งให้ข้อคิด ความรู้ที่มีประโยชน์ที่ใช้ประกอบการทำงานวิจัยจนสำเร็จลุล่วง

ขอขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร.กฤษณะ ชินสาร รองศาสตราจารย์ ดร.จีรพร วีระพันธุ์ ผู้ช่วยศาสตราจารย์ ดร.นวลสวาท หิรัญสกลวงศ์ และผู้ช่วยศาสตราจารย์ ดร.กุลสวัสดิ์ จิตขจรวานิช คณะกรรมการสอบวิทยานิพนธ์ ที่กรุณาให้คำแนะนำตลอดจนข้อชี้แนะจนในที่สุดทำให้วิทยานิพนธ์ฉบับนี้สำเร็จลงได้

ขอขอบคุณพระคุณบิดา มารดา และพี่ๆ ที่สนับสนุนให้ได้เรียนในระดับที่ได้ตั้งใจ อีกทั้งยังได้สนับสนุนดูแลเรื่องค่าใช้จ่ายต่างๆ ระหว่างการศึกษาเป็นอย่างดีอีกด้วย

ท้ายสุดขอขอบคุณเพื่อนๆ และพี่น้องทุกคนที่ให้คำปรึกษา และช่วยอำนวยความสะดวกในด้านต่างๆ

สำหรับคุณงามความดีและประโยชน์อันใดที่เกิดจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับบิดา มารดา อาจารย์ทุกท่านซึ่งเป็นที่เคารพรักยิ่ง ตลอดจนญาติพี่น้อง รุ่นพี่และเพื่อนๆ ที่รักทุกคน

นางสาวบุญหทัย เครือแก้ว

สารบัญ

	หน้า
บทคัดย่อภาษาไทย	ก
บทคัดย่อภาษาอังกฤษ	ข
กิตติกรรมประกาศ	ค
สารบัญ	ง
สารบัญตาราง	ช
สารบัญรูป	ฌ
บทที่ 1 บทนำ	1
1.1 ความเป็นมาและความสำคัญของงานปัญหา	1
1.2 วัตถุประสงค์ของงานวิจัย	3
1.3 สมมติฐานของงานวิจัย	3
1.4 ขอบเขตของงานวิจัย	4
1.5 ประโยชน์ที่คาดว่าจะได้รับ	4
บทที่ 2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง	5
2.1 การประมวลผลแบบกลุ่มเมฆ	5
2.1.1 หลักการของการประมวลผลแบบกลุ่มเมฆ	5
2.1.2 โครงสร้างของการประมวลผลแบบกลุ่มเมฆ	7
2.1.3 การให้บริการของระบบการประมวลผลแบบกลุ่มเมฆ	8
2.1.4 ประเภทของการประมวลผลแบบกลุ่มเมฆ	10
2.1.5 คุณลักษณะเบื้องต้นของการประมวลผลแบบกลุ่มเมฆ	12
2.1.6 ข้อดีและข้อจำกัดของการประมวลผลแบบกลุ่มเมฆ	13
2.2 เทคโนโลยีการจำลองเครื่องคอมพิวเตอร์เสมือน	14
2.2.1 ตัวอย่างของผลิตภัณฑ์ที่ใช้ในการจำลองเทคโนโลยีเครื่องคอมพิวเตอร์เสมือน	14
2.2.2 ส่วนประกอบของสถาปัตยกรรมคอมพิวเตอร์แบบเสมือน	15
2.2.3 ประโยชน์ที่ได้รับจากการจำลองเทคโนโลยีเครื่องคอมพิวเตอร์เสมือน	16
2.3 ปัญญาประดิษฐ์	16
2.4 วิธีการเรียนรู้แบบเสริมกำลัง	17
2.5 ขั้นตอนวิธีอาณานิคมผึ้งเทียม	21
2.6 การหาค่าความเหมาะสมที่สุดด้วยวิธีอาณานิคมมด	24
2.7 ขั้นตอนวิธีการหาคำตอบเหมาะสมที่สุดแบบกลุ่มอนุภาค	26

สารบัญ (ต่อ)

	หน้า
2.8 ขั้นตอนวิธีการหาค่าเหมาะสมที่สุดแบบนกกาเหว่า	28
2.9 ปัญหาการหาค่าเหมาะสมที่มีหลายวัตถุประสงค์	30
2.9.1 หลักการพื้นฐานของการหาค่าเหมาะสมที่สุด	30
2.9.2 ลักษณะของปัญหาการหาค่าเหมาะสมที่สุดที่มีหลายวัตถุประสงค์	31
2.9.3 วิธีสำหรับการหาค่าเหมาะสมที่สุดที่มีหลายวัตถุประสงค์	31
2.10 เครื่องมือที่ใช้ในการสร้างแบบจำลอง CloudSim-3.0.3	32
2.10.1 ลักษณะทั่วไปของ CloudSim	33
2.10.2 คุณสมบัติทั่วไปของ CloudSim	33
2.10.3 รูปแบบการทำงานของ CloudSim	34
2.11 งานวิจัยที่เกี่ยวข้อง	35
บทที่ 3 วิธีการดำเนินงานวิจัย	42
3.1 นิยามปัญหาและการกำหนดฟังก์ชันวัตถุประสงค์	42
3.2 การประยุกต์ใช้ขั้นตอนอาณานิคมผึ้งเทียม	47
3.3 การประยุกต์ใช้ขั้นตอนอาณานิคมผึ้งเทียมร่วมกับการเรียนรู้แบบเสริมแรง	50
3.3.1 ขั้นตอนวิธีการเรียนรู้แบบคิว	51
3.3.2 การประยุกต์ใช้ขั้นตอนอาณานิคมผึ้งเทียม	53
บทที่ 4 การทดลองและผลการทดลอง	56
4.1 ข้อมูลที่ใช้ในการทดสอบประสิทธิภาพ	56
4.2 การเตรียมข้อมูล	57
4.3 เครื่องมือที่ใช้ในงานวิจัย	58
4.4 ผลการทดลองเพื่อหาพารามิเตอร์ที่เหมาะสมของขั้นตอนอาณานิคมผึ้งเทียม	58
4.4.1 การตั้งค่าพารามิเตอร์สภาพแวดล้อมในการทดลอง	58
4.4.2 การตั้งค่าพารามิเตอร์ในขั้นตอนวิธีอาณานิคมผึ้งเทียม	59
4.4.3 ผลการทดสอบประสิทธิภาพ เมื่อจำนวนผึ้งในระบบมีการเปลี่ยนแปลง	59
4.5 ผลการทดลองเพื่อประเมินประสิทธิภาพของการจัดตารางงานโดยใช้ขั้นตอนอาณานิคมผึ้งเทียมควบคู่กับการจัดตารางงานแบบฮิวริสติก	65
4.5.1 การตั้งค่าพารามิเตอร์สภาพแวดล้อมในการทดลอง	65
4.5.1.1 กรณีสภาพแวดล้อมภายในเครื่องคอมพิวเตอร์เสมือนเหมือนกัน	65
4.5.1.2 กรณีสภาพแวดล้อมภายในเครื่องคอมพิวเตอร์เสมือนแตกต่างกัน	66

สารบัญ (ต่อ)

หน้า

4.5.2 การตั้งค่าพารามิเตอร์ใน HABC, ACO, PSO และ IPSO 67
4.5.3 ผลการทดลองเพื่อประเมินประสิทธิภาพในการทำงาน 68
4.5.3.1 ผลการทดสอบประสิทธิภาพในส่วนของเวลาการทำงานรวมเฉลี่ย 68
4.5.3.2 ผลการทดสอบประสิทธิภาพในแง่ของค่าความไม่สมดุลของการกระจายข้อมูล 75
4.5.3.3 ผลการทดสอบประสิทธิภาพในส่วนของค่าเบี่ยงเบนมาตรฐาน 77
4.6 ผลการทดลองเพื่อประเมินประสิทธิภาพของการจัดตารางงานโดยใช้ MOABCQ 78
4.6.1 การตั้งค่าพารามิเตอร์สภาพแวดล้อมในการทดลอง 79
4.6.2 การตั้งค่าพารามิเตอร์สำหรับอัลกอริทึมเมตาฮิวริสติกที่ใช้ในการเปรียบเทียบ 80
4.6.3 ชุดข้อมูลที่ใช้ในการทดลอง 81
4.6.4 ผลการทดลองเพื่อประเมินประสิทธิภาพในการทำงาน 81
4.6.4.1 ผลการทดสอบประสิทธิภาพในกรณีเวลาทำงานรวม 82
4.6.4.2 ผลการทดสอบประสิทธิภาพในกรณีปริมาณงานต่อหน่วยเวลา 84
4.6.4.3 ผลการทดสอบประสิทธิภาพในกรณีอัตราส่วนการใช้ทรัพยากรโดยเฉลี่ย 85
4.6.4.4 ผลการทดสอบประสิทธิภาพในกรณีค่าความไม่สมดุลของการกระจาย 86
4.6.4.5 ผลการทดสอบประสิทธิภาพในกรณีค่าใช้จ่าย 87
4.7 ค่าความซับซ้อนของวิธี MOABCQ 89
บทที่ 5 การทดลองและผลการทดลอง 90
5.1 สรุปผลและวิเคราะห์ผลการทดลอง 90
5.2 ข้อจำกัด 91
5.2 ข้อเสนอแนะ 93
เอกสารอ้างอิง 94
ภาคผนวก 103
ภาคผนวก งานวิจัยที่ตีพิมพ์ 104
ประวัติผู้เขียน 135

สารบัญตาราง

ตารางที่ **หน้า**

4.1 ชุดข้อมูลที่ใช้ในการทดสอบประสิทธิภาพ 57

4.2 ตัวอย่างในการจัดเรียงข้อมูล 57

4.3 ค่าพารามิเตอร์ของแบบจำลองการประมวลผลแบบกลุ่มเมฆ กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมแตกต่างกันทั้งหมด 58

4.4 ค่าพารามิเตอร์ของขั้นตอนวิธีอาณานิคมผึ้งเทียม 59

4.5 ชุดข้อมูลที่ใช้ในการทดสอบประสิทธิภาพของการทดสอบกรณีที่ 4.4.3 60

4.6 เวลาในการทำงานรวมโดยใช้ชุดข้อมูลประเภทแบบสุ่มอิสระ 62

4.7 เวลาในการทำงานรวมโดยใช้ชุดข้อมูลประเภทแบบการกระจายข้อมูลแบบแจกแจงปกติ 63

4.8 ค่าพารามิเตอร์ของแบบจำลองการประมวลผลแบบกลุ่มเมฆ กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมเหมือนกัน 65

4.9 ค่าพารามิเตอร์ของแบบจำลองการประมวลผลแบบกลุ่มเมฆ กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมแตกต่างกัน (HABC) 66

4.10 ค่าของพารามิเตอร์ที่ใช้ในวิธีการที่นำเสนอและอัลกอริทึมอื่นที่นำมาเปรียบเทียบ 68

4.11 เวลาการทำงานรวมของชุดข้อมูลที่ D_1 – D_3 เมื่อเปรียบเทียบระหว่าง HABC, ACO, PSO และ IPSO (Homogenous) 69

4.12 เวลาการทำงานรวมของชุดข้อมูลที่ D_1 – D_3 เมื่อเปรียบเทียบระหว่าง HABC, ACO, PSO และ IPSO (Heterogenous) 70

4.13 เวลาการทำงานรวมของชุดข้อมูลที่ D_4 – D_6 เมื่อเปรียบเทียบระหว่าง HABC, ACO, PSO และ IPSO (Homogenous) 71

4.14 เวลาการทำงานรวมของชุดข้อมูลที่ D_4 – D_6 เมื่อเปรียบเทียบระหว่าง HABC, ACO, PSO และ IPSO (Heterogenous) 71

4.15 เวลาการทำงานรวมของชุดข้อมูลที่ D_7 – D_9 เมื่อเปรียบเทียบระหว่าง HABC, ACO, PSO และ IPSO (Homogenous) 72

4.16 เวลาการทำงานรวมของชุดข้อมูลที่ D_7 – D_9 เมื่อเปรียบเทียบระหว่าง HABC, ACO, PSO และ IPSO (Heterogenous) 73

4.17 เวลาการทำงานรวมของชุดข้อมูลที่ D_{10} – D_{12} เมื่อเปรียบเทียบระหว่าง HABC, ACO, PSO และ IPSO (Homogenous) 74

4.18 เวลาการทำงานรวมของชุดข้อมูลที่ D_{10} – D_{12} เมื่อเปรียบเทียบระหว่าง HABC, ACO, PSO และ IPSO (Heterogenous) 74

สารบัญตาราง (ต่อ)

ตารางที่ **หน้า**

4.19 ค่าส่วนเบี่ยงเบนมาตรฐานกรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในเหมือนกัน 77
4.20 ค่าส่วนเบี่ยงเบนมาตรฐานกรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในแตกต่างกัน77
4.21 ค่าพารามิเตอร์ของแบบจำลองสภาพแวดล้อมในการทดลอง 79
4.22 ค่าพารามิเตอร์ของอัลกอริทึมเมตาฮิวริสติกที่ใช้ในการเปรียบเทียบ 80
4.23 เปรียบเทียบประสิทธิภาพของ MOABCQ ในแง่อัตราส่วนการใช้ทรัพยากรโดยเฉลี่ย 85

สารบัญรูป

รูปที่		หน้า
2.1	หลักการทำงานของระบบการประมวลผลแบบกลุ่มเมฆ	5
2.2	สถาปัตยกรรมการให้บริการบนการประมวลผลแบบกลุ่มเมฆ	8
2.3	ระบบการประมวลผลแบบกลุ่มเมฆแบ่งตามขอบเขตการใช้งาน	10
2.4	สถาปัตยกรรมของเทคโนโลยีการจำลองเครื่องคอมพิวเตอร์เสมือนด้วยซอฟต์แวร์	15
2.5	ส่วนประกอบของปัญญาประดิษฐ์	17
2.6	ประเภทของการเรียนรู้ของเครื่อง	18
2.7	วงรอบการทำงานของวิธีการเรียนรู้แบบเสริมกำลัง	18
2.8	การเต้นแกว่งของผึ้งเพื่อสื่อสารข้อมูลของแหล่งอาหาร	22
2.9	ตัวอย่างลักษณะการเคลื่อนที่แบบ Lévy Flights	28
2.10	วิธีผลรวมถ่วงน้ำหนัก	32
2.11	วิธีพื้นฐานพาเรโต	32
2.12	สถาปัตยกรรมลำดับชั้นของ CloudSim	33
2.13	การทำงานของ CloudSim	34
3.1	ผังงานแสดงการจัดตารางงานเพื่อเข้าใช้เครื่องคอมพิวเตอร์เสมือนและระบบมีการกระจายงานแบบสมดุล โดยประยุกต์ใช้ขั้นตอนวิธีอาณานิคมผึ้งเทียม	47
3.2	กระบวนการขั้นตอนวิธีการเรียนรู้แบบคิว	51
4.1	เวลาทำงานรวมเฉลี่ยของชุดข้อมูล D_1 เมื่อกำหนดค่าพารามิเตอร์ของ ABC โดยใช้ Case I	60
4.2	เวลาทำงานรวมเฉลี่ยของชุดข้อมูล D_1 เมื่อกำหนดค่าพารามิเตอร์ของ ABC โดยใช้ Case II	61
4.3	เวลาทำงานรวมเฉลี่ยของชุดข้อมูล D_1 เมื่อกำหนดค่าพารามิเตอร์ของ ABC โดยใช้ Case III	61
4.4	เวลาทำงานรวมเฉลี่ยของชุดข้อมูล D_4 เมื่อกำหนดค่าพารามิเตอร์ของ ABC โดยใช้ Case I	64
4.5	เวลาทำงานรวมเฉลี่ยของชุดข้อมูล D_4 เมื่อกำหนดค่าพารามิเตอร์ของ ABC โดยใช้ Case II	64
4.6	เวลาทำงานรวมเฉลี่ยของชุดข้อมูล D_4 เมื่อกำหนดค่าพารามิเตอร์ของ ABC โดยใช้ Case III	64
4.7	ค่าความไม่สมดุลของการกระจายข้อมูล กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในเหมือนกัน	76
4.8	ค่าความไม่สมดุลของการกระจายข้อมูล กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในแตกต่างกัน	76
4.9	เปรียบเทียบประสิทธิภาพของ MOABCQ ในแง่ของเวลทำงานรวม	83
4.10	เปรียบเทียบประสิทธิภาพของ MOABCQ ในแง่ของปริมาณงานต่อหน่วยเวลา	84
4.11	เปรียบเทียบประสิทธิภาพของ MOABCQ ในแง่ของค่าใช้จ่าย	88

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ปัจจุบันการติดต่อสื่อสารผ่านอินเทอร์เน็ตถือเป็นสิ่งจำเป็นในชีวิตประจำวัน อินเทอร์เน็ตกลายเป็นศูนย์กลางในการเชื่อมโยงข้อมูลต่างๆ เข้าด้วยกัน เนื่องจากผู้ใช้งานมีปริมาณเพิ่มมากขึ้น ประกอบกับข้อมูลที่มีการเปลี่ยนแปลงตลอดเวลา ทำให้ภาระในการจัดการทำงานของคอมพิวเตอร์มีปริมาณเพิ่มมากขึ้นเช่นกัน ซึ่งบางครั้งอาจส่งผลให้ไม่สามารถตอบสนองความต้องการของผู้ใช้ได้อย่างทั่วถึง จากปัญหาดังกล่าวจึงได้มีการนำเทคโนโลยีการให้บริการการประมวลผลแบบกลุ่มเมฆ (Cloud computing) มาใช้เพื่อมุ่งเน้นการให้บริการทางเครือข่ายและคอมพิวเตอร์ (Network and computing) การจัดเก็บข้อมูล (Storage) และการให้บริการทรัพยากรข้อมูล (Data service resource) ไว้ด้วยกัน

การประมวลผลแบบกลุ่มเมฆเริ่มต้นจากการที่ต้องการให้ผู้ใช้สามารถเข้าถึงทรัพยากรการประมวลผลหรือสามารถซื้อบริการคลาวด์ (Cloud service) ได้ตามความต้องการ โดยมีแนวคิดแบบการใช้ทรัพยากรร่วมกันเมื่อต้องการ (On-demand resource sharing) ผ่านการใช้งานบนอินเทอร์เน็ต อีกทั้งยังสามารถประมวลผลได้หลากหลายขึ้นอยู่กับบริการและช่องทางการให้บริการ (Working platforms) และความต้องการของผู้ใช้งาน [1] การประมวลผลแบบกลุ่มเมฆเป็นการรวมการทำงานแบบกระจาย (Distributed computing) และกระประมวลผลแบบขนาน (Parallel computing) ที่มีการใช้ทรัพยากรร่วมกัน เช่น ซอฟต์แวร์ (Software) ฮาร์ดแวร์ (Hardware) เป็นต้น มีรูปแบบการใช้งานแบบ “Pay-as-you-go” [2] ซึ่งในการใช้งานผู้ใช้ไม่จำเป็นต้องซื้อซอฟต์แวร์ใดๆ เพียงแต่มีการเชื่อมต่ออินเทอร์เน็ต เพื่อเข้าใช้งานและชำระเงินเท่าที่มีการใช้งานเท่านั้น

บริการโครงสร้างพื้นฐานคลาวด์ (Infrastructure as a Service: IaaS) เป็นหนึ่งในบริการของคลาวด์ที่ให้บริการเครื่องคอมพิวเตอร์เสมือน (Virtual machine: VM) หรือเครื่องแม่ข่าย (Server) สำหรับประมวลผลและการจัดเก็บข้อมูลในระบบคลาวด์ ผู้ใช้สามารถเรียกใช้ระบบปฏิบัติการหรือแอปพลิเคชัน (Application) ใดๆ บนเครื่องแม่ข่ายที่เช่าโดยไม่เสียค่าบำรุงรักษาของเครื่องแม่ข่ายที่เป็นฮาร์ดแวร์ เนื่องจากโครงสร้างพื้นฐานของระบบคลาวด์ มีการทำงานอยู่บนรูปแบบเทคโนโลยีเสมือน (Virtualization technology) ส่งผลให้ระบบไม่ถูกจำกัดในเรื่องของขีดความสามารถในการประมวลผลในระบบ IaaS ยังมีข้อดีอื่น ได้แก่ การให้ผู้ใช้สามารถเข้าถึงเครื่องแม่ข่ายในสถานที่ที่ใกล้เคียงกับผู้ใช้ปลายทาง ทั้งนี้ขึ้นอยู่กับความต้องการของผู้ใช้ (Quality of service: QoS) และข้อตกลงระดับการบริการ (Service level agreement: SLA) ในส่วนของการชำระค่าบริการนั้นขึ้นอยู่กับข้อตกลงระหว่างผู้ใช้และผู้ให้บริการคลาวด์ (Cloud service provider: CSP) [3]

เครื่องคอมพิวเตอร์เสมือนมีการใช้ทรัพยากรของเครื่องโฮสต์ (Host) เช่น แรม (Random Access Memory: Ram) ฮาร์ดดิสก์ (Hard disk) หรือหน่วยประมวลผลกลาง (Central Processing Unit: CPU) ซึ่งทรัพยากรที่อยู่ในเครื่องคอมพิวเตอร์เสมือน นั้นขึ้นอยู่กับความต้องการที่มีการร้องขอเข้าใช้งานทรัพยากรสำหรับใช้ในการปฏิบัติงาน ด้วยเหตุนี้ในการประมวลผลแบบกลุ่มเมฆจึงมีการกระจายของทรัพยากรที่ไม่เท่ากัน เครื่องคอมพิวเตอร์เสมือนบางเครื่องอาจไม่ได้รับการจัดสรรทรัพยากรตามความต้องการ เนื่องจากเครื่องคอมพิวเตอร์เสมือนจำนวนมากมีการเชื่อมต่อกับทรัพยากรเป็นแบบหน่วยประมวลผลกลางสามารถสลับจากสถานะการทำงานเป็นสถานะพร้อมหรือสถานะรอเป็นสถานะพร้อม (Preemptive) และเป็นแบบที่หน่วยประมวลผลกลางไม่มีการสลับสถานการณ์ทำงานคือ การกำหนดเวลาเกิดขึ้นเมื่อกระบวนการยุติหรือสลับจากสถานะการทำงานเพื่อรอสถานะของการตั้งเวลาจากหน่วยประมวลผลกลาง (Non-preemptive) [4]

ในกรณีที่มีการร้องขอเพื่อเข้าใช้บริการไปยังเครื่องแม่ข่ายเดียวกัน ในบางครั้งเครื่องแม่ข่ายตัวอื่นๆ ไม่มีการขอเข้าใช้บริการ ซึ่งเรียกได้ว่าเกิดการกระจายงานแบบไม่สมดุลในระบบ (System load imbalance) ซึ่งสามารถแก้ไขได้โดยการทำการจัดตารางการทำงาน (Task scheduling) ก่อนเข้าใช้บริการ เพื่อให้เกิดการกระจายการทำงานแบบสมดุลในระบบ (System load balance) ซึ่งการจัดตารางการทำงานที่ดีจะมีประโยชน์ในแง่ของการเพิ่มประสิทธิภาพการใช้งานทรัพยากรมากที่สุด คือ เป็นการลดการสูญเสียเวลาในการรอเข้าทำงานในระบบให้น้อยที่สุด อีกทั้งยังสามารถลดค่าใช้จ่ายที่เกิดขึ้นจากการเข้าใช้งานอีกด้วย

การจัดตารางเข้าทำงานเพื่อเข้าใช้ทรัพยากรในระบบการประมวลผลแบบกลุ่มเมฆ เพื่อให้ระบบเกิดการกระจายงานแบบสมดุลนั้น จำเป็นต้องคำนึงถึงเงื่อนไขต่างๆ เช่น ความเสถียรของระบบ (Stability) ความรวดเร็วในการทำงาน (Processing time) ค่าใช้จ่าย (Cost) การใช้พลังงาน (Energy consumption) เวลารอเข้าทำงาน (Waiting time) การใช้ทรัพยากร (Resource utilization) ความปลอดภัย (Security) เวลารวมในการทำงานของระบบ (Makespan) และความน่าเชื่อถือ (Reliability) [5] เป็นต้น ซึ่งจากเงื่อนไขที่ต้องคำนึงถึงการจัดตารางงานนั้นจะมีลักษณะคล้ายคลึงกับการทำงานของขั้นตอนอาณานิคมผึ้งเทียม (Artificial bee colony algorithm: ABC algorithm) ในส่วนของพฤติกรรมการออกหาอาหารของผึ้ง

ขั้นตอนอาณานิคมผึ้งเทียมเป็นวิธีการเมตาฮิวริสติก (Meta-heuristic method) ที่ใช้ในการแก้ปัญหาหาคำตอบของวิธีการต่างๆ เพื่อให้ได้คำตอบที่ใกล้เคียงค่าที่เหมาะสม ที่พัฒนาขึ้นโดย Karaboga [6,7] ขั้นตอนอาณานิคมผึ้งเทียมได้เลียนแบบพฤติกรรมการออกหาอาหารของอาณานิคมผึ้ง ซึ่งต้องมีการปรับตัวตามสภาพแวดล้อมของที่อยู่อาศัยและแหล่งอาหาร มีงานวิจัยหลายชิ้นแสดงให้เห็นถึงประสิทธิภาพของขั้นตอนอาณานิคมผึ้งเทียมในการปัญหาต่างๆ เช่น ปัญหาการเดินทางของพนักงานขาย (Traveling salesman problem) [8] ปัญหาการจัดตารางการผลิต (Job shop scheduling problem) [9] เป็นต้น ขั้นตอนอาณานิคมผึ้งเทียมมีรูปแบบการทำงานที่คล้ายกับวิธีการเรียนรู้แบบเสริมกำลัง (Reinforcement learning) [10] ที่มีการเรียนรู้ตามสภาพแวดล้อมเป็นการ

เรียนรู้ที่เกิดจากสิ่งที่สังเกตได้เพื่อนำมาใช้ในการทำนายหรือตัดสินใจคำตอบ ตัวแทนจะเรียนรู้วิธีการปฏิบัติตนในสภาพแวดล้อมโดยดำเนินการและเห็นผลลัพธ์ของตัวเอง ในที่นี้ผู้วิจัยนำขั้นตอนวิธีการเรียนรู้แบบคิว (Q-learning algorithm) ซึ่งเป็นหนึ่งในวิธีการเรียนรู้แบบเสริมกำลังเข้ามาช่วยในการทำนายและตัดสินใจในการเลือกจัดตารางงานที่เหมาะสม

ดังนั้นในงานวิจัยนี้จึงได้เสนอวิธีการจัดตารางงานที่เป็นอิสระต่อกัน (Independent task) ในการประมวลผลแบบกลุ่มเมฆ โดยมุ่งเน้นที่วิธีการจัดตารางงานแบบที่มีหลายวัตถุประสงค์ (Multi-objective optimization scheduling method) โดยใช้ขั้นตอนอาณานิคมผึ้งเทียมควบคู่กับการปรับใช้ขั้นตอนวิธีการเรียนรู้แบบคิว เพื่อช่วยให้การใช้ขั้นตอนอาณานิคมผึ้งเทียมสามารถทำงานได้เร็วขึ้น เรียกอัลกอริทึมนี้ว่า “MOABCQ” ซึ่งอัลกอริทึมที่นำเสนอนี้มีวัตถุประสงค์เพื่อเพิ่มประสิทธิภาพการใช้งานของทรัพยากรของระบบ และให้มีการกระจายงานระหว่างเครื่องคอมพิวเตอร์เสมือนแบบสมดุล (Load balancing) โดยพิจารณาจากเวลารวมทั้งหมดในการทำงาน (Makespan) ค่าใช้จ่ายที่ใช้ในการเข้าทำงานในเครื่องคอมพิวเตอร์เสมือนของงาน และการใช้งานทรัพยากรพร้อมกันน้อยที่สุด (Utilization of resources) เป็นข้อจำกัดของการพิจารณาพร้อมกัน

1.2 วัตถุประสงค์ของงานวิจัย

1) เพื่อเพิ่มประสิทธิภาพการทำงานในการประมวลผลแบบกลุ่มเมฆ

2) เพื่อนำเสนอขั้นตอนวิธีการจัดตารางเวลาเพื่อเข้าใช้ทรัพยากรเครื่องคอมพิวเตอร์เสมือนที่มีสภาพแวดล้อมแตกต่างกัน (Heterogeneous environment) ในการประมวลผลแบบกลุ่มเมฆ

3) ประยุกต์ใช้ขั้นตอนอาณานิคมผึ้งเทียมควบคู่กับวิธีการเรียนรู้แบบเสริมกำลัง เพื่อแก้ปัญหาในการจัดสรรการใช้งานทรัพยากร (Resource allocation) โดยพิจารณาวิธีการจัดตารางงานแบบที่มีหลายวัตถุประสงค์

4) ลดเวลารวมในการทำงานของระบบ ลดค่าใช้จ่าย และเพิ่มประสิทธิภาพการเข้าใช้งานทรัพยากรในระบบ

5) จัดลำดับการเข้าใช้เครื่องคอมพิวเตอร์เสมือน เพื่อให้มีการกระจายการทำงานในระบบแบบสมดุล

1.3 สมมติฐานของงานวิจัย

การใช้ขั้นตอนวิธีอาณานิคมผึ้งเทียมควบคู่กับวิธีการเรียนรู้แบบเสริมกำลัง ในการจัดตารางงานเพื่อเข้าใช้ทรัพยากรเครื่องคอมพิวเตอร์เสมือนที่มีสภาพแวดล้อมแตกต่างกัน ในการประมวลผลแบบกลุ่มเมฆ ซึ่งคำนึงถึงวิธีการจัดตารางงานแบบที่มีหลายวัตถุประสงค์ โดยพิจารณาจากค่าใช้จ่ายที่เกิดขึ้น เวลารวมในการทำงาน และประสิทธิภาพของการเข้าใช้ทรัพยากรในระบบ เพื่อให้ระบบมีการกระจายงานที่สมดุล อีกทั้งยังสามารถช่วยลดเวลารวมในการเข้าทำงาน ลดค่าใช้จ่ายที่เกิดขึ้นจากการเข้าใช้งานและสามารถใช้งานทรัพยากรที่มีอยู่ในระบบอย่างมีประสิทธิภาพอีกด้วย

1.4 ขอบเขตของงานวิจัย

1) จำลองระบบการประมวลผลแบบกลุ่มเมฆเป็นแบบเฉพาะที่ (Private cloud) โดยใช้เครื่องมือในการจำลองที่เรียกว่า CloudSim-3.0.3

2) กำหนดรูปแบบการใช้หน่วยประมวลกลางเป็นแบบที่ไม่สามารถขัดจังหวะหรือแทรกกลางคันได้ (Non-preemptive process)

3) ทำการศึกษาการจัดตารางการทำงานภายใต้สมมติฐานที่ว่าเครื่องจักรมีการเสีย (Down time) ระหว่างดำเนินการ

4) เครื่องคอมพิวเตอร์เสมือนทุกเครื่องอยู่ในสถานะพร้อมทำงาน ไม่มีค้างในสถานะรอทำงานและเครื่องเริ่มทำงาน ณ เวลาที่ 0 วินาที

5) งานวิจัยนี้ได้เสนอขั้นตอนวิธีการจัดตารางเข้าทำงานโดยการจัดตารางงานแบบที่มีหลายวัตถุประสงค์ เพื่อเข้าใช้ทรัพยากรเครื่องคอมพิวเตอร์เสมือนที่มีสภาพแวดล้อมแตกต่างกัน โดยการใช้ขั้นตอนวิธีอาณานิคมผึ้งเทียมควบคู่กับวิธีการเรียนรู้แบบเสริมกำลัง

6) ชุดข้อมูลที่ใช้ในการทดสอบมี 4 รูปแบบ คือ ชุดข้อมูลแบบสุ่มอิสระ (Random) ชุดข้อมูลที่มีการกระจายแบบโค้งปกติ (Symmetric distribution) ชุดข้อมูลที่มีการกระจายตัวแบบเบ้ขวา (Right-skewed distribution) และชุดข้อมูลที่มีการกระจายตัวแบบเบ้ซ้าย (Left-skewed distribution)

7) ชุดข้อมูลที่ใช้ในการทดสอบมี 3 ชุดข้อมูลดังนี้ คือ ชุดข้อมูลแบบสุ่มอิสระ ชุดข้อมูล GoCJ (Google Cloud Jobs: GoCJ) [11] และชุดข้อมูล Synthetic workload [12]

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1) สามารถจัดตารางเวลางานเพื่อเข้าใช้เครื่องคอมพิวเตอร์เสมือนในระบบการประมวลผลแบบกลุ่มเมฆได้

2) ทราบถึงขั้นตอนวิธีการประยุกต์ใช้ขั้นตอนอาณานิคมผึ้งเทียมควบคู่กับวิธีการเรียนรู้แบบเสริมแรง โดยใช้ขั้นตอนวิธีการเรียนรู้แบบคิว เพื่อใช้ในการจัดตารางงานเข้าทำงานและจัดสรรทรัพยากรให้เหมาะสมในการประมวลผลแบบกลุ่มเมฆ

3) สามารถลดเวลารวมในการเข้าทำงาน และเพิ่มประสิทธิภาพการใช้งานของทรัพยากร

4) สามารถประยุกต์ใช้ขั้นตอนอาณานิคมผึ้งเทียมควบคู่กับวิธีการเรียนรู้แบบเสริมแรง เพื่อให้ระบบสามารถกระจายงานได้แบบสมดุล

5) ได้แบบจำลองที่ช่วยในการวางแผนการเข้าใช้เครื่องคอมพิวเตอร์เสมือนในการประมวลผลแบบกลุ่มเมฆ

6) สามารถประเมินสถานการณ์ของการเข้าใช้เครื่องคอมพิวเตอร์เสมือนให้ทราบล่วงหน้าได้

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในงานวิจัยนี้เสนอการประยุกต์ใช้ปัญญาประดิษฐ์สำหรับปัญหาการจัดตารางงานที่มีหลายวัตถุประสงค์ในการประมวลผลแบบกลุ่มเมฆ ได้มีการศึกษาทฤษฎีและงานวิจัยที่เกี่ยวข้อง โดยได้แบ่งเนื้อหาออกเป็น 11 หัวข้อย่อย ได้แก่ การประมวลผลแบบกลุ่มเมฆ เทคโนโลยีการจำลองเครื่องคอมพิวเตอร์เสมือน ปัญญาประดิษฐ์ วิธีการเรียนรู้แบบเสริมกำลัง ขั้นตอนอาณานิคมผึ้งเทียม การหาค่าความเหมาะสมที่สุดด้วยวิธีอาณานิคมมด ขั้นตอนวิธีการหาคำตอบเหมาะสมที่สุดแบบกลุ่มอนุภาค ขั้นตอนวิธีการหาค่าเหมาะสมที่สุดแบบนกกาเหว่า ปัญหาการหาค่าเหมาะสมที่มีหลายวัตถุประสงค์ เครื่องมือที่ใช้ในการสร้างแบบจำลอง CloudSim-3.0.3 และงานวิจัยที่เกี่ยวข้อง

2.1 การประมวลผลแบบกลุ่มเมฆ

2.1.1 หลักการของการประมวลผลแบบกลุ่มเมฆ

การประมวลผลแบบกลุ่มเมฆ (Cloud computing) [13,14] คือ ระบบคอมพิวเตอร์ที่มีการประมวลผลอยู่ในระบบอินเทอร์เน็ต บนรูปแบบของโครงสร้างการประมวลผลขนาดใหญ่ที่ทำงานร่วมกัน มีการแบ่งปันทรัพยากรในการประมวลผลร่วมกันบนเครือข่ายอินเทอร์เน็ต ราวกับว่าเป็นหน่วยประมวลผลเดียวกัน

รูปที่ 2.1 หลักการทำงานของระบบการประมวลผลแบบกลุ่มเมฆ [13]

รูปที่ 2.1 แสดงถึงตัวอย่างการใช้งานระบบการประมวลผลแบบกลุ่มเมฆ ซึ่งการประมวลผลแบบกลุ่มเมฆจะมีรูปแบบการประมวลผลที่อิงกับความต้องการของผู้ใช้ โดยผู้ใช้สามารถระบุความต้องการไปยังซอฟต์แวร์ของระบบการประมวลผลแบบกลุ่มเมฆ จากนั้นซอฟต์แวร์จะร้องขอให้ระบบจัดสรรทรัพยากรและบริการให้ตรงกับความต้องการของผู้ใช้งาน ทั้งนี้ระบบสามารถเพิ่มหรือลดจำนวนของทรัพยากร รวมถึงเสนอบริการให้เหมาะสมกับความต้องการของผู้ใช้ได้ตลอดเวลา โดยที่ผู้ใช้ไม่จำเป็นต้องทราบการทำงานหรือเหตุการณ์เบื้องหลัง

การประมวลผลแบบกลุ่มเมฆเป็นเทคโนโลยีที่พัฒนาขึ้นมาเพื่อตอบสนองรูปแบบการทำงานและการใช้งานระบบสารสนเทศผ่านเครือข่ายอินเทอร์เน็ต โดยเน้นไปที่การจัดทำระบบประมวลผลและโครงสร้างพื้นฐานคอมพิวเตอร์ขนาดใหญ่ เพื่อรองรับการใช้งานของผู้ใช้จำนวนมากผ่านโปรแกรมประยุกต์ที่ทำงานแบบบริการด้านซอฟต์แวร์ (Software as a Service: SaaS) ผู้ใช้งานการประมวลผลแบบกลุ่มเมฆไม่จำเป็นต้องรับภาระการดำเนินการจัดทำระบบคอมพิวเตอร์หรือการขยายระบบเมื่อองค์กรมีจำนวนผู้ใช้มากขึ้น ซึ่งจะเป็นการลดต้นทุนในส่วนของคอมพิวเตอร์ประมวลผลและค่าบำรุงรักษา แนวคิดของระบบการประมวลผลแบบกลุ่มเมฆนี้เปรียบเสมือนกับการบริการไฟฟ้า โดยมองการบริการประมวลผลด้วยคอมพิวเตอร์เป็นเหมือนโครงสร้างสาธารณูปโภคพื้นฐานที่องค์กรหรือหน่วยงานสามารถใช้งานได้โดยไม่มีขอบเขต [15-16]

นิยามความหมายของคำหลักๆ 3 คำที่เกี่ยวข้องกับการประมวลผลแบบกลุ่มเมฆมีดังนี้

1. ความต้องการ (Requirement) คือ โจทย์ปัญหาที่ผู้ใช้ต้องการให้ระบบคอมพิวเตอร์แก้ไขปัญหาหรือตอบปัญหาตามที่ผู้ใช้กำหนดได้ ยกตัวอย่างเช่น ความต้องการพื้นที่จัดเก็บข้อมูลขนาด 1,000,000 GB เป็นต้น

2. ทรัพยากร (Resource) หมายถึง ปัจจัยหรือสิ่งที่เกี่ยวข้องกับการประมวลผล หรือเกี่ยวข้องกับการแก้ไขปัญหาตามโจทย์ที่ความต้องการของผู้ใช้ได้ระบุไว้ เช่น หน่วยประมวลผลกลาง หน่วยความจำ เช่น แรม อุปกรณ์จัดเก็บข้อมูล เช่น ฮาร์ดดิสก์ ฐานข้อมูล (Database) สารสนเทศ (Information) ข้อมูล (Data) เครือข่าย (Network) ซอฟต์แวร์ประยุกต์ (Application software) เซ็นเซอร์ที่สามารถรับส่งสัญญาณอินพุทและเอาต์พุท (Remote sensor) เป็นต้น

3. บริการ (Service) ถือว่าเป็นทรัพยากรและในทางกลับกันก็สามารถบอกได้ว่า ทรัพยากรก็คือบริการ โดยเฉพาะอย่างยิ่งในด้านการประมวลผลแบบกลุ่มเมฆแล้ว สามารถจะใช้คำว่าบริการแทนคำว่าทรัพยากร คำว่าบริการหมายถึงการกระทำ (Operation) เพื่อให้เกิดผลลัพธ์ที่สนองต่อความต้องการ แต่การกระทำของบริการจะเกิดขึ้นได้จำเป็นต้องพึ่งพาทรัพยากรที่เกี่ยวข้องเพื่อแก้ปัญหาให้เกิดผลลัพธ์สนองต่อความต้องการ

ระบบการประมวลผลแบบกลุ่มเมฆ [16] ได้รับความสนใจเป็นอย่างมาก เนื่องจากความสามารถของการขยายหรือลดขนาดของระบบ และความยืดหยุ่นโดยปราศจากการลงทุนในเรื่องของการติดตั้งและการดำเนินงานในส่วนของการวางโครงสร้างพื้นฐานทางด้านเทคโนโลยีสารสนเทศ

และศูนย์รวมข้อมูล กระบวนการของระบบประมวลผลกลุ่มเมฆนี้คาดว่าจะทำให้องค์กรสามารถขยายหรือลดขนาดของระบบในส่วนของเทคโนโลยีพื้นฐานขององค์กรได้ตามความต้องการ

2.1.2 โครงสร้างของการประมวลผลแบบกลุ่มเมฆ

สถาปัตยกรรมในมุมมองการตลาดของการประมวลผลแบบกลุ่มเมฆประกอบไปด้วยองค์ประกอบพื้นฐาน 4 ส่วน [13] ดังนี้

1. ผู้ใช้ (User/Broker) คือ ผู้ที่ทำการส่งคำร้องขอใช้บริการซึ่งจะถูกส่งมาจากที่ใดก็ได้

2. ข้อตกลงในการจัดสรรทรัพยากรในการให้บริการของผู้ให้บริการ (Service level agreement (SLA) Resource allocator) ที่เปรียบเสมือนส่วนเชื่อมประสาน (Interface) ระหว่างผู้บริการประมวลผลในกลุ่มเมฆกับผู้ใช้บริการที่อยู่นอกกลุ่มเมฆ การทำงานของข้อตกลงในการจัดสรรทรัพยากรในการให้บริการของผู้ให้บริการ ประกอบไปด้วย 6 ส่วนย่อย ดังนี้

1) ส่วนที่ทำหน้าที่แปลคำขอบริการ (Service request examiner and admission control) ทำหน้าที่แปลความหมายของคำร้องขอใช้บริการที่ถูกส่งมาจากผู้ใช้บริการ คำร้องขอใช้บริการจะถูกนำไปเปรียบเทียบกับคุณภาพของบริการ ซึ่งเป็นข้อตกลงในการใช้บริการที่ได้กระทำไว้ล่วงหน้า เพื่อพิจารณาว่า คำร้องขอใช้บริการจะได้รับการยอมรับหรือปฏิเสธ รวมทั้งการจัดสรรทรัพยากรประมวลผลอย่างมีประสิทธิภาพโดยไม่โหลดมากเกินไป จากนั้นจึงส่งคำร้องขอใช้บริการไปยังเครื่องคอมพิวเตอร์เสมือนและกำหนดเครื่องสำหรับการจัดการเครื่องคอมพิวเตอร์เสมือน

2) ส่วนคำนวณค่าใช้จ่าย (Pricing) กลไกในส่วนนี้มีไว้เพื่อคำนวณค่าใช้จ่ายในการใช้บริการตามเงื่อนไขของการใช้บริการ เช่น เวลาที่ใช้คำร้องขอใช้บริการเป็นช่วงเวลาสูงสุดหรือไม่ อัตราค่าบริการเป็นอัตราคงที่หรือผันแปร เป็นต้น

3) ส่วนติดตามการใช้บริการ (Accounting) เป็นกลไกสำหรับใช้บรรจุข้อมูลบนร่องของจานบันทึก (Track) การใช้ทรัพยากรที่ถูกร้องขอและมีการคิดค่าใช้จ่ายกับผู้ใช้บริการ กลไกนี้จะมีหน้าที่บันทึกประวัติการใช้งานทรัพยากรต่างๆ เพื่อนำไปปรับปรุงการจัดสรรทรัพยากรในการทำทรัพยากรเสมือน

4) ส่วนตรวจติดตามระบบเสมือน (Virtual machine monitor) เป็นกลไกที่เก็บบันทึกข้อมูลบนร่องของจานบันทึกของเครื่องคอมพิวเตอร์เสมือนที่จัดให้พร้อมใช้งานรวมทั้งทรัพยากรที่นำมาจัดเป็นเครื่องคอมพิวเตอร์เสมือน

5) ส่วนจ่ายงาน (Dispatcher) เป็นโปรแกรมเลือกจ่ายงานซึ่งถูกใช้เป็นกลไกในการเริ่มต้นการทำงานตามคำร้องของบริการที่ได้รับการยอมรับให้ทำงานบนเครื่องเสมือนที่จัดสรรไว้กับงานนั้นๆ

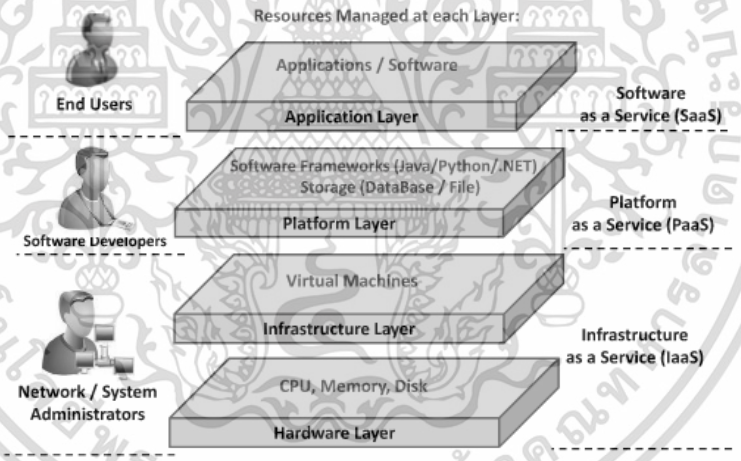
6) ส่วนบันทึกการดำเนินการ (Service request monitor) เป็นกลไกในการเก็บรักษาข้อมูลบนร่องของจานบันทึกของความก้าวหน้าในการดำเนินงานตามคำร้องขอบริการ

3. เครื่องคอมพิวเตอร์เสมือน (VM) เครื่องคอมพิวเตอร์ที่มีอยู่จริงหนึ่งเครื่องสามารถจัดเป็นเครื่องเสมือนได้หลายเครื่อง ทั้งนี้ขึ้นอยู่กับระดับคำร้องขอบริการ เครื่องคอมพิวเตอร์เสมือนหลายๆ เครื่องสามารถดำเนินงานตามชุดคำสั่งงานได้พร้อมกันบนเครื่องคอมพิวเตอร์ที่มีอยู่จริงเพียงเครื่องเดียว ทั้งนี้เพราะว่าเครื่องคอมพิวเตอร์เสมือนแต่ละเครื่องจะทำงานเป็นอิสระจากกันแม้จะอยู่บนเครื่องคอมพิวเตอร์ที่มีอยู่จริงเครื่องเดียวกัน

4. เครื่องคอมพิวเตอร์ที่มีอยู่จริง (Physical machines) ประกอบไปด้วยเครื่องแม่ข่ายจำนวนมากที่เตรียมไว้สำหรับให้บริการ

2.1.3 การให้บริการของระบบการประมวลผลแบบกลุ่มเมฆ

รูปแบบการให้บริการของการประมวลผลแบบกลุ่มเมฆ สามารถแบ่งได้เป็น 3 กลุ่ม ได้แก่ บริการด้านโครงสร้างฮาร์ดแวร์พื้นฐาน (Infrastructure-as-a-Service: IaaS) บริการด้านแพลตฟอร์ม (Platform-as-a-Service: PaaS) และบริการด้านซอฟต์แวร์ (Software-as-a-Service: SaaS) ดังแสดงในรูปที่ 2.2 ซึ่งประมวลผลตามความต้องการของผู้ใช้ ทำให้สามารถเรียกใช้งานได้ง่ายและสะดวก



รูปที่ 2.2 สถาปัตยกรรมการให้บริการบนการประมวลผลแบบกลุ่มเมฆ [13]

รายละเอียดรูปแบบการให้บริการของการประมวลผลแบบกลุ่มเมฆสามารถอธิบายได้ดังต่อไปนี้

1. บริการด้านโครงสร้างฮาร์ดแวร์พื้นฐาน

บริการด้านโครงสร้างฮาร์ดแวร์พื้นฐานเป็นบริการที่เกี่ยวกับโครงสร้างพื้นฐานของระบบคอมพิวเตอร์ผ่านอินเทอร์เน็ต โดยผู้ใช้บริการโครงสร้างพื้นฐาน (Infrastructure Provider: IP) เป็นผู้จัดหาทรัพยากรสำหรับประมวลผล ไม่ว่าจะเป็นส่วนของการจัดเก็บ หรือประมวลผลผ่านระบบเสมือนที่ผู้ใช้บริการสามารถกำหนดขนาดทรัพยากรให้ตรงกับความต้องการหรือสามารถเปลี่ยนแปลง

บริการให้ตรงกับความต้องการของผู้ใช้งาน ในระดับโครงสร้างพื้นฐานเหมาะสมกับองค์กรที่ไม่ต้องการลงทุนทางด้านฮาร์ดแวร์ซึ่งก็คือ การให้บริการในด้านต่างๆ ดังนี้

1) ทรัพยากรต่างๆ ในรูปของบริการ เช่น เครื่องแม่ข่าย หน่วยความจำ หน่วยประมวลผลกลาง พื้นที่เก็บข้อมูลบนฮาร์ดดิสก์ หรืออุปกรณ์เครือข่าย เป็นต้น

2) การขยายขนาดของโครงสร้างพื้นฐานซึ่งสามารถทำให้เล็กหรือใหญ่ได้ขึ้นอยู่กับความต้องการของแอปพลิเคชัน

3) การกำหนดราคาในการให้บริการนั้นจะขึ้นอยู่กับการเลือกใช้บริการทรัพยากร

ความสามารถในการขยายหรือลดขนาดของระบบ (Scalability) คือคุณสมบัติที่โดดเด่นของระบบประมวลผลกลุ่มเมฆ บริการในระดับ IaaS นั้นจะตอบสนองความต้องการในเรื่องของทรัพยากรด้านพื้นที่ในการเก็บข้อมูล เช่น Amazon's Simple Storage Service และความยืดหยุ่นของการใช้ทรัพยากร เช่น GoGrid, RightScale, Linode และ Amazon Compute Elastic (EC2) เป็นต้น

2. บริการด้านแพลตฟอร์ม

บริการด้านแพลตฟอร์มสำหรับการพัฒนาซอฟต์แวร์และแอปพลิเคชัน โดยผู้ให้บริการจะจัดเตรียมสิ่งที่จำเป็นต้องใช้ในการพัฒนาซอฟต์แวร์และแอปพลิเคชัน เช่น เว็บแอปพลิเคชัน เครื่องคอมพิวเตอร์แม่ข่ายสำหรับงานจัดเก็บข้อมูล (Database server) ระบบประมวลผลกลางสำหรับองค์กรขนาดใหญ่ และซอฟต์แวร์กลางที่เชื่อมต่อระหว่างผู้ใช้ (Client) และเครื่องแม่ข่ายเข้าด้วยกัน หรือที่เรียกว่า Middleware เป็นต้น โดยบริการทั้งหมดทำงานภายใต้ระบบรักษาความปลอดภัยเครือข่าย และสามารถเรียกใช้งานได้ผ่าน เว็บแอปพลิเคชัน ซึ่งผู้ใช้บริการสามารถนำไปปรับใช้และจัดการได้เอง บริการด้านแพลตฟอร์มนั้นประกอบด้วย ระบบปฏิบัติการ ระบบฐานข้อมูล และระบบมิดเดิลแวร์ ตัวอย่างเช่น Window Server, Linux, Oracle Database เป็นต้น ตัวอย่างผู้ให้บริการในระดับแพลตฟอร์ม เช่น Google Apps, Heroku Mosso และ Engine Yard เป็นต้น

ข้อดีของบริการด้านแพลตฟอร์ม คือ มีความพร้อมในการใช้งานและไม่ต้องกังวลกับฮาร์ดแวร์หรือการปรับเพิ่มลดขนาดของระบบตามจำนวนผู้ใช้ที่เพิ่มมากขึ้น สามารถสร้างแอปพลิเคชัน สร้างบริการหรือโซลูชั่นใหม่ๆ ได้โดยไม่จำเป็นต้องเน้นนักพัฒนาที่มีทักษะสูงในการบำรุงรักษาซอฟต์แวร์

ข้อจำกัดของบริการด้านแพลตฟอร์ม คือ ถ้าผู้ใช้พัฒนาโปรแกรมโดยผูกติดกับส่วนต่อประสานโปรแกรมประยุกต์ (Application programming interface: API) ของผู้ให้บริการ PaaS รายใดรายหนึ่ง หากโปรแกรมมีความซับซ้อนมากก็ยากที่จะย้ายไปใช้งานที่อื่นได้ และส่วนใหญ่ PaaS จะไม่ค่อยเปิดให้ใช้ฟรี

3. บริการด้านซอฟต์แวร์

การให้บริการด้านแอปพลิเคชันซึ่งใครก็สามารถเข้ามาใช้ได้ โดยที่ผู้ใช้ไม่จำเป็นต้องเป็นเจ้าของ และไม่จำเป็นต้องรู้ว่าแอปพลิเคชันนั้นถูกติดตั้งที่ใด หรือมีการใช้ทรัพยากรอะไรบ้าง ทาง

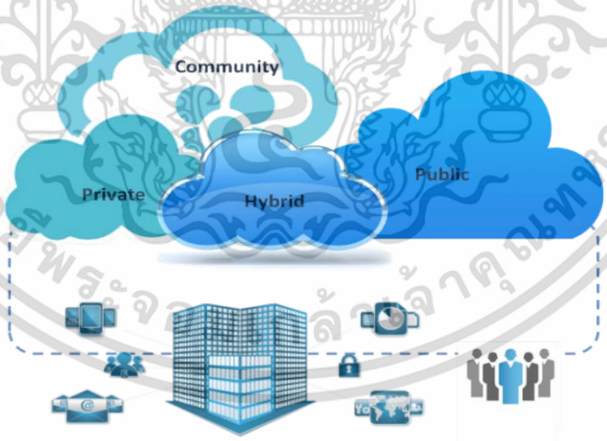
ผู้ให้บริการจะเป็นผู้จัดเตรียมทั้งหมด ผู้ใช้นั้นเพียงแค่เข้าไปใช้งานเท่านั้น และผู้ให้บริการก็จะคิดค่าบริการตามการใช้งานจริงของผู้ใช้แอปพลิเคชัน ในระดับแอปพลิเคชันนั้นมีซอฟต์แวร์ ให้เลือกใช้อย่างมากมาย โดยไม่ต้องใช้เงินลงทุนจำนวนมาก นั่นคือการให้บริการ ซอฟต์แวร์ในรูปแบบเซอร์วิสเล็กๆ ที่ทำหน้าที่เฉพาะทาง ทำงานตามฟังก์ชันที่กำหนดไว้ ไม่ได้ทำงานเหมือนแอปพลิเคชันใหญ่ๆ ที่มีความสามารถมากมายรวมอยู่ในตัวเดียว โดยการใช้งานนั้นผู้ใช้ SaaS เองไม่จำเป็นต้องเป็นเจ้าของเซอร์วิสนั้น เพราะจะสามารถเลือกใช้บริการเซอร์วิสของผู้ให้บริการใดก็ได้ในลักษณะของการเช่า หรือสมัครเป็นสมาชิกเพื่อใช้งานเซอร์วิสนั้นๆ ผู้ให้บริการมีหน้าที่บริการและสร้างเซอร์วิสใหม่ๆ ขึ้นมาและดูแลระบบต่างๆ ให้สามารถให้บริการตัวเซอร์วิสนั้นได้ตามความต้องการของลูกค้า การให้บริการแอปพลิเคชันต่างๆ เช่น Google Apps หรือ การให้บริการ E-mail เป็นต้น

ข้อดีของ SaaS คือ ใช้ต้นทุนต่ำ เพราะไม่ต้องเสียค่าใช้จ่ายไปกับการติดตั้งบำรุงรักษาพัฒนาระบบ สามารถใช้งานได้ทุกที่เพียงแค่มีการเชื่อมต่ออินเทอร์เน็ต ข้อมูลไม่เสียหายและสามารถจัดเก็บได้ง่ายขึ้น เพราะถูกรวมอยู่ในศูนย์กลางข้อมูลเดียวกัน ฯลฯ

ข้อจำกัดของ SaaS คือ ปัญหาด้านความปลอดภัยของข้อมูล

2.1.4 ประเภทของการประมวลผลแบบกลุ่มเมฆ

การประมวลผลแบบกลุ่มเมฆ [17,18] สามารถแบ่งตามขอบเขตการใช้งาน แสดงได้ดังรูปที่ 2.3



รูปที่ 2.3 ระบบการประมวลผลแบบกลุ่มเมฆแบ่งตามขอบเขตการใช้งาน [18]

รูปแบบการประมวลผลแบบกลุ่มเมฆสามารถแบ่งออกเป็น 3 ประเภท ดังนี้

1. การประมวลผลแบบกลุ่มเมฆแบบส่วนตัว (Private cloud)

บริการของผู้ให้บริการที่นำการประมวลผลแบบกลุ่มเมฆไปทำงานบนเครือข่ายส่วนบุคคล (Private network) ของผู้ให้บริการหรือเครือข่ายที่เปิดให้ผู้ใช้บริการเฉพาะราย บนหลักการ

ของเทคโนโลยีการจำลองเครื่องคอมพิวเตอร์เสมือน (Virtualization technology) ผู้ใช้บริการสามารถควบคุมและจัดการระบบได้ด้วยตนเอง การประมวลผลแบบกลุ่มเมฆส่วนตัว สามารถแก้ปัญหาในเรื่องของความเสถียรและความน่าเชื่อถือในการใช้งานระบบได้

จุดเด่นของการประมวลผลแบบกลุ่มเมฆส่วนตัว ได้แก่

1) สามารถควบคุมระบบการเข้าถึงและความปลอดภัยได้อย่างอิสระ

2) ข้อมูลมีความปลอดภัย เนื่องจากระบบฮาร์ดแวร์และซอฟต์แวร์มีความเป็นส่วนตัวสูง

3) ควบคุมต้นทุนการบริการและการดูแลระบบได้ชัดเจนกว่า เนื่องจากผู้ใช้งานสามารถกำหนดการบริหารจัดการได้ชัดเจนมากกว่า

4) ลดปัญหาการเชื่อมต่อและลดความล่าช้า (Latency) หรือระยะเวลาการตอบสนองลงได้มากกว่า โดยแบนด์วิธ (Bandwidth) ในการใช้งานก็มีความเป็นส่วนตัวอีกด้วย

5) องค์กรสามารถบริหารจัดการทรัพยากรทั้งหมดได้ด้วยตนเองผ่านการควบคุมบนแพลตฟอร์ม

2. การประมวลผลแบบกลุ่มเมฆแบบสาธารณะ (Public cloud)

การประมวลผลแบบกลุ่มเมฆแบบสาธารณะเป็นระบบการประมวลผลหรือการจัดเก็บข้อมูลของผู้ใช้งานที่ถูกจัดเก็บบนเครื่องแม่ข่ายสาธารณะ ซึ่งผู้ให้บริการจะมีการเก็บข้อมูลของผู้ใช้งานแบบสาธารณะไว้ในเครื่องแม่ข่ายเดียวกัน แต่ผู้ใช้งานจะไม่สามารถเข้าถึงข้อมูลของบุคคลอื่นได้ หากไม่ได้รับการอนุญาต ตามหลักการของการประมวลผลแบบกลุ่มเมฆที่จัดให้มีการใช้ทรัพยากรร่วมกัน (Share resource) ในศูนย์กลางข้อมูล โดยที่ผู้ใช้บริการสามารถระบุความต้องการได้ด้วยตัวเองและจ่ายค่าใช้บริการตามปริมาณการใช้งานจริง เช่น การประมวลผลแบบกลุ่มเมฆของ Amazon เป็นต้น

จุดเด่นของการประมวลผลแบบกลุ่มเมฆแบบสาธารณะ ได้แก่

1) ลดต้นทุนค่าใช้จ่ายเช่น การติดตั้ง ดูแลและการบำรุงรักษา เป็นต้น

2) สามารถเพิ่มหรือลดทรัพยากรการใช้งานได้ตามที่ต้องการ ผ่านการสั่งงานบนแพลตฟอร์ม

3) ลดระยะเวลาเครื่องหยุดทำงาน (Downtime) หรือปัญหาระบบล่ม เนื่องจากมีมาตรฐานการรับรองจากผู้ให้บริการ เช่น ISO/IEC, SLA เป็นต้น

4) ใช้งานง่าย เนื่องจากมีการออกแบบเพื่อให้ควบคุมการทำงานที่สะดวกและมีการสอนใช้งาน รวมถึงมีเจ้าหน้าที่คอมพิวเตอร์ช่วยเหลือ

3. การประมวลผลแบบกลุ่มเมฆแบบผสม (Hybrid cloud)

การนำเอารูปแบบของการประมวลผลแบบกลุ่มเมฆแบบส่วนตัวและการประมวลผลแบบกลุ่มเมฆแบบสาธารณะมาผสมผสานกัน นั่นคือการใช้ทรัพยากรที่มีอยู่มาสร้างเป็นกลุ่มเมฆขึ้นเพื่อใช้ภายในองค์กร แต่เนื่องจากทรัพยากรที่มีอยู่นั้นไม่เพียงพอกับระบบทั้งหมด จึงได้มีการใช้

บริการของกลุ่มเมฆแบบสาธารณะมาร่วมด้วยซึ่งจะทำให้สามารถเพิ่มทรัพยากรขั้นพื้นฐานได้เพียงพอกับระบบที่จะนำมาใช้ จุดเด่นของการประมวลผลแบบกลุ่มเมฆแบบผสม คือ ลดต้นทุนในการจัดการน้อยลงเนื่องจากไม่ต้องลงทุนกับการประมวลผลแบบกลุ่มเมฆแบบส่วนตัวทั้งหมด

2.1.5 คุณลักษณะเบื้องต้นของการประมวลผลแบบกลุ่มเมฆ

ลักษณะเบื้องต้นที่เป็นปัจจัยสำคัญของการประมวลผลแบบกลุ่มเมฆ [15] มี 5 ลักษณะ ดังนี้

1. บริการตนเองตามความต้องการ (On demand self service) คือ ผู้ใช้งานสามารถกำหนดประสิทธิภาพของการประมวลผลได้ด้วยตัวเอง เช่น ขนาดของหน่วยความจำในเครือข่าย ความเร็วและจำนวนของหน่วยประมวลผลกลางหรือ สามารถกำหนดความสามารถต่างๆ ให้ยืดหยุ่นโดยอัตโนมัติโดยไม่ต้องมีคนดูแล

2. การเข้าถึงได้หลายช่องทาง (Broad network access) คือ ความสามาถในการให้บริการผ่านระบบเครือข่ายและสามารถเข้าถึงได้โดยผ่านมาตรฐานกลางไม่ว่าจะอยู่แพลตฟอร์มใด เช่น โทรศัพท์มือถือ หรือเครื่องคอมพิวเตอร์ เป็นต้น โดยระบบต้องสามารถรองรับการใช้งานบนแพลตฟอร์มที่มีความหลากหลาย (Multi-platform)

3. การใช้ทรัพยากรร่วมกัน (Resource pooling) คือ ผู้ให้บริการจะจัดหาทรัพยากรในการประมวลผลเพื่อที่จะให้บริการกับผู้ใช้งานจำนวนมาก โดยใช้ในรูปแบบของการให้เช่าทรัพยากรที่หลากหลายตามความต้องการของผู้ใช้ ผู้ใช้สามารถกำหนดขอบเขตของสมรรถนะที่ต้องการโดยไม่ต้องมีความรู้ด้านการควบคุมบริหารจัดการหรือรู้ที่อยู่ของอุปกรณ์ประมวลผลของผู้ใช้งานเลย ตัวอย่างของอุปกรณ์ต่างๆ เช่น ฮาร์ดดิสก์ หน่วยประมวลผลกลาง หน่วยความจำ เป็นต้น

4. มีความยืดหยุ่นในการให้บริการสูง (Rapid elasticity) ระบบมีความสามารถของการยืดหยุ่น สามารถตรวจสอบได้และมีความรวดเร็วในการดำเนินงาน ในบางกรณีสามารถทำได้โดยอัตโนมัติเพื่อขยายขนาดของการประมวลผลของระบบให้เพียงพอต่อความต้องการ ซึ่งผู้ใช้งานสามารถตกลงกับผู้ให้บริการในเรื่องของการจัดเตรียมสมรรถนะของการประมวลผล เช่น เรื่องของเวลาในการทำงาน หรือการควบคุมปริมาณการประมวลผล เป็นต้น

5. ระบบการวัดบริการ (Measured service) ระบบคลาวด์โดยปกติและจะมีการควบคุมและจัดสรรทรัพยากรแบบอัตโนมัติ โดยใช้การวัดปริมาณการใช้งานของอุปกรณ์ที่ใช้งานในแต่ละประเภทของการให้บริการ เช่น ฮาร์ดดิสก์ หน่วยประมวลผลกลาง หน่วยความจำ ระบบเครือข่ายหรือจำนวนผู้เข้าใช้งาน ดังนั้น ทรัพยากรต่างๆ เหล่านี้จึงควรที่จะสามารถมีการควบคุมดูแลและสามารถออกรายงานได้เพื่อเป็นส่วนประกอบในการตัดสินใจของผู้ใช้งานในการพิจารณาการเพิ่มหรือลดขนาดของระบบ

2.1.6 ข้อดีและข้อจำกัดของการประมวลผลแบบกลุ่มเมฆ

ข้อดีของการประมวลผลแบบกลุ่มเมฆ [13,14] มีหลายประการดังนี้

1. ประหยัดค่าใช้จ่าย (Cost savings) มีค่าใช้จ่ายที่น้อยกว่า เนื่องจากผู้ใช้ไม่ต้องลงทุนในเรื่องของทรัพยากรหรือดูแลระบบ เพราะผู้ให้บริการจะเป็นฝ่ายลงทุนทรัพยากรด้านเทคโนโลยีเกือบทั้งหมด อีกทั้งการประมวลผลแบบกลุ่มเมฆนั้นยังมีบริการที่จ่ายตามการใช้งาน (Pay as you go) ซึ่งช่วยประหยัดค่าใช้จ่ายได้มาก

2. ความคล่องตัว (Agility) เพิ่มความสะดวกและความรวดเร็วในการใช้งาน เนื่องจากในการใช้งานนั้นผู้ใช้งานไม่ต้องคำนึงถึงเรื่องของสถานที่ เวลา หรืออุปกรณ์ แต่ขอเพียงแค่ผู้ใช้มีอุปกรณ์ที่สามารถเชื่อมต่ออินเทอร์เน็ตได้ ผู้ใช้ก็สามารถเข้าถึงและใช้งานคลาวด์ได้แบบไม่มีข้อจำกัด

3. ความสามารถในการปรับขนาดและมีความยืดหยุ่น (Scalability and Flexibility) มีความสามารถในการขยายตัวได้อย่างรวดเร็ว เนื่องจากความสามารถของการประมวลผลแบบกลุ่มเมฆนั้น ช่วยให้ผู้ใช้สามารถสร้างระบบใหม่ได้ทุกเวลา ทั้งยังสามารถเพิ่มหรือลดขนาดของทรัพยากรได้ในระยะเวลาสั้น

4. สะดวก (Mobility) การเข้าถึงคลาวด์มีความสะดวกมากเพียงแค่มีอินเทอร์เน็ต ซึ่งผู้ให้บริการคลาวด์ จะมี SLA ให้ระบบสามารถเข้าถึงได้ตลอด และมีการกระจายโหนด (Node) ของระบบในประเทศต่างๆ เพื่อกระจายความเสี่ยง

5. ข้อมูลเชิงลึก (Insight) ระบบควบคุมอัตโนมัติของคลาวด์ช่วยให้ง่ายต่อการวิเคราะห์การเข้าถึงระบบเชิงลึกได้

6. การควบคุมคุณภาพ (Quality control) ในหน้าจอควบคุมของระบบคลาวด์จะมีรายการของการใช้งาน และระบบต่างๆ ที่อยู่ในคลาวด์ ทำให้การควบคุมคุณภาพของระบบ IT รวมศูนย์อยู่ที่เดียว ก่อให้เกิดการควบคุมคุณภาพได้อย่างสะดวกมากยิ่งขึ้น

7. การกู้คืนระบบ (Disaster recovery) การกู้คืนบนคลาวด์ทำได้ง่ายและสะดวก โดยสามารถเลือกให้สำรองข้อมูลแบบอัตโนมัติ (Auto-Backup) หรือกู้คืนข้อมูลด้วยตนเอง (Manual recovery) ได้ และหากมีเหตุการณ์ระบบล่มหรือเกิดข้อผิดพลาดจากระบบคลาวด์ ทางผู้ให้บริการจะมีการประกันความเสี่ยง และสามารถกู้คืนระบบให้ทันที ภายในระยะเวลาตามข้อกำหนด

8.อัพเดตซอฟต์แวร์อัตโนมัติ (Automatic software updates) เข้าถึงนวัตกรรมใหม่ๆ อยู่เสมอ เนื่องจากผู้ให้บริการ คลาวด์จะทำหน้าที่พัฒนาและอัพเกรดระบบใหม่ๆ รวมถึงสรรหาเทคโนโลยีที่ทันสมัยเพื่อรองรับการใช้งานของผู้ใช้บริการ เช่น เครื่องคอมพิวเตอร์เสมือน (VM) บล็อกเชน (Blockchain) หรือโปรแกรมประยุกต์ใหม่ๆ เพื่อรองรับการทำงานและตอบโจทย์ของผู้ใช้ได้มากที่สุด โดยผู้ใช้ไม่ต้องหาบริการใหม่ๆเอง

9. ความปลอดภัย (Security) มีการบำรุงรักษาและความปลอดภัยสูง ผู้ให้บริการมีมาตรการการรักษาความปลอดภัยให้กับโครงสร้างพื้นฐานของศูนย์กลางข้อมูล และการเก็บรักษา

ข้อมูลของลูกค้า ดังนั้นข้อมูลของผู้ใช้จะถูกเก็บไว้ภายใต้ระบบรักษาความปลอดภัยที่รัดกุม อีกทั้งยังมีทีมงานผู้เชี่ยวชาญคอยดูแลรักษาและซ่อมแซมระบบตลอด 24 ชั่วโมง

สำหรับข้อจำกัดของการประมวลผลแบบกลุ่มเมฆ มีหลายประการดังนี้

1) การที่มีทรัพยากรจากหลายแห่งอาจเกิดปัญหาด้านความต่อเนื่องและความรวดเร็ว

2) ไม่มีการรับประกันในการทำงานอย่างต่อเนื่องของระบบและความปลอดภัยของข้อมูล

3) แพลตฟอร์มยังไม่ได้มาตรฐานทำให้มีข้อจำกัดสำหรับตัวเลือกในการพัฒนาหรือติดตั้ง

2.2 เทคโนโลยีการจำลองเครื่องคอมพิวเตอร์เสมือน (Virtualization technology)

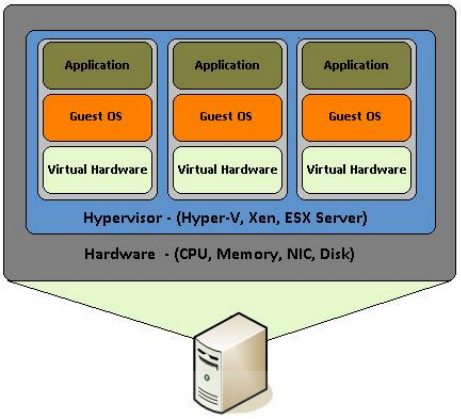
ปัจจุบันเทคโนโลยีทางด้านฮาร์ดแวร์นั้นก้าวหน้าไปมาก ในส่วนของประสิทธิภาพและความเร็วในการประมวลผล ทำให้เกิดแนวการสร้างระบบจำลองเสมือนจริงขึ้นที่เรียกว่า เทคโนโลยีการจำลองเครื่องคอมพิวเตอร์เสมือน เทคโนโลยีนี้ใช้สำหรับสร้างทรัพยากรเสมือนหรือทรัพยากรแบบนามธรรมของระบบคอมพิวเตอร์ เช่น สร้างหน่วยประมวลผลกลาง (CPU) การ์ดแสดงผล (Video card) การ์ดแลน (Local area network card: LAN card) เสมือน ซึ่งสิ่งที่ถูกสร้างขึ้นทั้งหมดนั้นเรียกว่า เครื่องคอมพิวเตอร์เสมือน [19, 20] โดยระบบเทคโนโลยีเสมือนจะควบคุมและจัดการทรัพยากรภายในระบบ โดยมีการแบ่งจัดสรรทรัพยากรให้กับแต่ละเครื่องคอมพิวเตอร์เสมือนอย่างเหมาะสมตามความต้องการ ดังนั้นแต่ละเครื่องคอมพิวเตอร์เสมือนจะมีองค์ประกอบเปรียบเสมือนเป็นเครื่องคอมพิวเตอร์ที่สมบูรณ์เครื่องหนึ่ง ทำให้สามารถรองรับระบบปฏิบัติการได้หลากหลายบนเครื่องแม่ข่ายหรือคอมพิวเตอร์เพียงเครื่องเดียว

2.2.1 ตัวอย่างของผลิตภัณฑ์ที่ใช้ในการจำลองเทคโนโลยีเครื่องคอมพิวเตอร์เสมือน

ตัวอย่างของผลิตภัณฑ์ที่ใช้ในการจำลองเทคโนโลยีเครื่องคอมพิวเตอร์เสมือน ได้แก่

1. ผลิตภัณฑ์จากบริษัท VMware ได้แก่ ESX, ESXi Server
2. ผลิตภัณฑ์จากบริษัท Microsoft ได้แก่ Hyper-V, Virtual Server
3. ผลิตภัณฑ์จากบริษัท Oracle ได้แก่ Oracle VM, VirtualBox
4. นอกเหนือจากนี้ก็จะมีผลิตภัณฑ์ของบริษัท Proxmox และ Eucalyptus ซึ่งเป็นซอฟต์แวร์ระบบเปิด (Open Source)

ซอฟต์แวร์เหล่านี้จะถูกเรียกว่า ไฮเปอร์ไวเซอร์ (Hypervisor) ซึ่งหมายถึงซอฟต์แวร์ที่ติดตั้งเพื่อใช้ในการจัดสรรทรัพยากรหรือเพื่อการสร้างเครื่องคอมพิวเตอร์เสมือน สถาปัตยกรรมของเทคโนโลยีการจำลองเครื่องคอมพิวเตอร์เสมือนด้วยซอฟต์แวร์แสดงดังรูปที่ 2.4



รูปที่ 2.4 สถาปัตยกรรมของเทคโนโลยีการจำลองเครื่องคอมพิวเตอร์เสมือนด้วยซอฟต์แวร์ [21]

สถาปัตยกรรมคอมพิวเตอร์แบบเสมือน (Virtualization) คือ การจำลองเครื่องคอมพิวเตอร์หลายๆ เครื่องขึ้นในโฮสต์ (Host) โดยจะเรียกเครื่องคอมพิวเตอร์ที่จำลองขึ้นมาว่า เกสต์ (Guest) และเกสต์จะต้องใช้ทรัพยากรของโฮสต์ร่วมกัน โดยเกสต์ที่สร้างขึ้นนั้นจะใช้ทรัพยากรได้มากที่สุดไม่เกินที่โฮสต์มี โดยมีไฮเปอร์ไวเซอร์เป็นตัวจัดการใช้ทรัพยากรของเกสต์ เกสต์จะมีการร้องขอทรัพยากรไปยังไฮเปอร์ไวเซอร์ และไฮเปอร์ไวเซอร์จะร้องขอไปยังระบบปฏิบัติการของโฮสต์ เพื่อป้องกันการใช้ทรัพยากรมากกว่าที่โฮสต์มี ซึ่งในปัจจุบันไฮเปอร์ไวเซอร์มีความสามารถมากขึ้น มีระบบจัดการไฮเปอร์ไวเซอร์แบบรวมศูนย์ ทำให้สามารถโยกย้ายเกสต์ข้ามโฮสต์ได้ ในกรณีที่โฮสต์มีปัญหาเพื่อรองรับการใช้งานได้ตลอดเวลา

2.2.2 ส่วนประกอบของสถาปัตยกรรมคอมพิวเตอร์แบบเสมือน

ส่วนประกอบของสถาปัตยกรรมคอมพิวเตอร์แบบเสมือน ได้แก่

1.ไฮเปอร์ไวเซอร์ เป็นระบบจัดการระหว่างโฮสต์และเกสต์ โดยทำหน้าที่ควบคุมทรัพยากรของโฮสต์ และการร้องขอทรัพยากรจากเกสต์ไปยังโฮสต์ ซึ่งไฮเปอร์ไวเซอร์จะควบคุมการจำลองของทรัพยากร ดังนี้ CPU, Memory, Storage, Network Interface Card (NIC) เป็นต้น

2. โฮสต์ คือ เครื่องคอมพิวเตอร์ที่ติดตั้งไฮเปอร์ไวเซอร์และเป็นเจ้าของทรัพยากร โดยไฮเปอร์ไวเซอร์ทำหน้าที่ร้องขอทรัพยากรที่ต้องการมายังระบบปฏิบัติการของโฮสต์ และระบบปฏิบัติการของ โฮสต์เป็นผู้ตอบสนอง

3. เกสต์ คือ เครื่องคอมพิวเตอร์ที่ถูกจำลองขึ้นจากโฮสต์โดยผ่านไฮเปอร์ไวเซอร์ โดยที่เกสต์นั้นจะมีสถาปัตยกรรมคอมพิวเตอร์ของไมโครโพรเซสเซอร์ (Microprocessor) ที่ล้อตามโฮสต์ เช่น X64, IA32, IA64 เป็นต้น การเลือกระบบปฏิบัติการที่ต้องการใช้ไม่สามารถติดตั้งข้ามสถาปัตยกรรมได้ และเกสต์จะมีทรัพยากรทุกอย่างเสมือนเป็นเครื่องคอมพิวเตอร์จริง เช่น CPU, Memory, Disk, NIC, IP address เป็นต้น

4. เครือข่ายของคอมพิวเตอร์แบบเสมือนนั้น ถูกออกมาเพื่อให้รองรับการสื่อสารที่มีข้อมูลมหาศาล เนื่องจากในหนึ่งโฮสต์จะมีมากกว่า 1 เกสต์ และเมื่อเกสต์ต้องการส่งหรือรับข้อมูลจากภายนอกจะต้องผ่าน NIC ของโฮสต์เสมอ ในปัจจุบันนี้นิยมใช้มาตรฐาน 1000BaseTx หรือ 1000BaseFx เป็นตัวกลางในการเชื่อมต่อระหว่างโฮสต์ และมีระบบที่ทำงานคู่กันเพื่อรองรับในกรณีที่อุปกรณ์เครือข่ายมีปัญหา

2.2.3 ประโยชน์ที่ได้รับจากการจำลองเทคโนโลยีเครื่องคอมพิวเตอร์เสมือน

ประโยชน์ที่ได้รับจากการจำลองเทคโนโลยีเครื่องคอมพิวเตอร์เสมือน ได้แก่

1. ล้มเลิกระบบเก่าที่กำหนดให้ 1 แอปพลิเคชันต้องทำงานอยู่บน 1 เครื่องแม่ข่าย โดยการนำเอางานของหลายเครื่องแม่ข่ายมาทำงานร่วมกันทำให้สามารถใช้งานเครื่องแม่ข่ายได้เต็มประสิทธิภาพมากขึ้น เช่น การรวมงานของเครื่องแม่ข่าย 2 เครื่องที่ใช้งาน CPU เพียงแค่ 40% ให้มาทำงานอยู่บนเครื่องแม่ข่ายเครื่องเดียว ซึ่งจะทำให้เครื่องแม่ข่ายอีกเครื่องนั้นว่าง และสามารถนำไปใช้กับงานอื่นได้มากขึ้น

2. ลดค่าใช้จ่ายของศูนย์รวมข้อมูลโดยลดจำนวนของโครงสร้างพื้นฐาน นั่นคือลดจำนวนของเครื่องแม่ข่ายลง ทำให้ต้องการพื้นที่น้อยลง ลดความจำเป็นในการใช้ไฟฟ้าและการทำความเย็น บริหารจัดการได้ง่ายขึ้นเนื่องจากมีจำนวนน้อย

3. ทำให้ฮาร์ดแวร์และซอฟต์แวร์นั้นว่างและอยู่ในสถานะพร้อมใช้งานมากขึ้น ส่งผลให้เกิดความต่อเนื่องในการดำเนินงาน

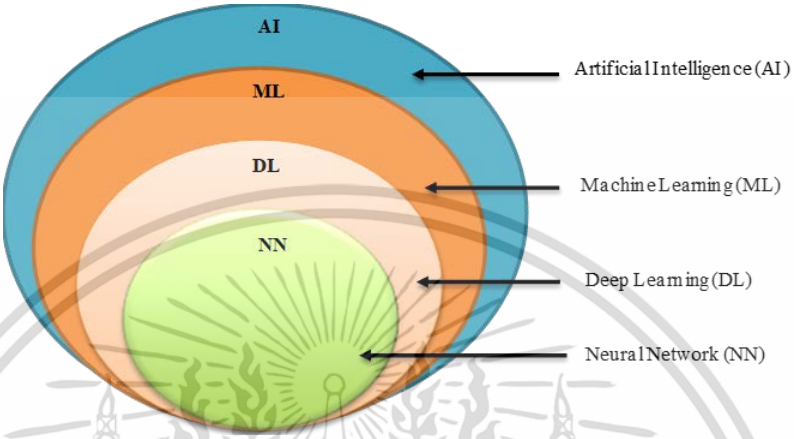
4. มีความยืดหยุ่นในการปฏิบัติงาน สามารถตอบสนองความต้องการทางธุรกิจได้แบบไดนามิก เพราะสามารถเพิ่มหรือลดทรัพยากรให้กับระบบงานที่ต้องการเฉพาะช่วงเวลาได้

2.3 ปัญญาประดิษฐ์

ปัญญาประดิษฐ์ (Artificial intelligence: AI) [22] เป็นแนวทางการพัฒนาทางด้านคอมพิวเตอร์อีกรูปแบบหนึ่ง ซึ่งทำให้เครื่องคอมพิวเตอร์คิดและตัดสินใจได้ใกล้เคียงกับมนุษย์ โดยอาศัยหลักการจากการศึกษาวิธีคิด การตัดสินใจหรือหลักของเหตุผลจากมนุษย์ เพื่อนำไปใช้ในการพัฒนาศักยภาพของเครื่องคอมพิวเตอร์ให้สามารถตอบสนองการทำงานที่มากกว่าเป็นเพียงเครื่องจักรกลหรือโปรแกรมทั่วไป โดยเริ่มจากการนำแนวคิดดังกล่าวมากำหนดเป็นขั้นตอนให้เครื่องคอมพิวเตอร์ทำงานแก้ปัญหา ตัดสิน และเรียนรู้ได้ด้วยตนเอง ส่งผลให้เครื่องคอมพิวเตอร์มีความฉลาดมากขึ้น สามารถทำงานในระบบที่มีความซับซ้อนได้อย่างมีประสิทธิภาพโดยไม่ต้องอาศัยแรงงานจากมนุษย์

รูปแบบการทำงานของปัญญาประดิษฐ์ เปรียบเสมือนวงกลมที่ซ้อนกัน 4 วง แสดงดังรูปที่ 2.5 จากรูปแสดงให้เห็นว่ามีจุดศูนย์กลางร่วมกัน ปัญญาประดิษฐ์คือวงใหญ่ที่สุดอยู่นอกสุด วงด้านในถัดมา

ก็คือ การเรียนรู้ของเครื่อง (Machine learning: ML) ML เป็นส่วนหนึ่งของ AI มีหน้าที่ในการเรียนรู้แต่ภายใต้การทำงาน ML นั้นก็ยังต้องขึ้นอยู่กับวงกลมถัดไปคือ การเรียนรู้เชิงลึก (Deep learning: DL) และสุดท้ายคือวงในสุด เป็นส่วนที่ทำหน้าที่สำคัญที่การทำงานของ ทั้ง 3 วงนอกจะต้องขึ้นอยู่กับวงนี้ก็คือ โครงข่ายประสาทเทียม (Neural networks: NN)

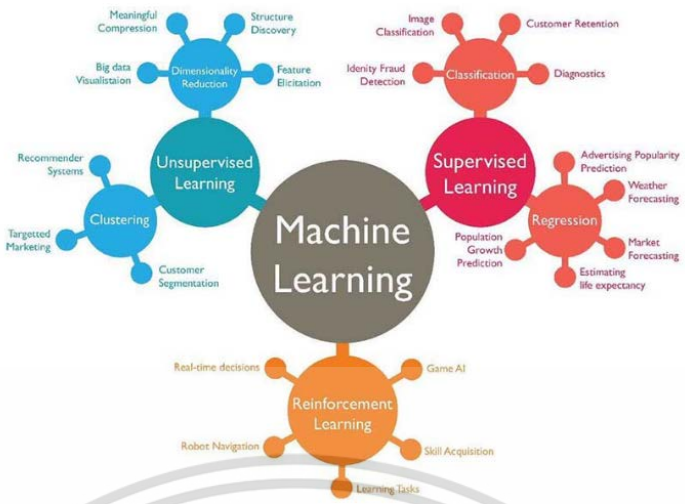


รูปที่ 2.5 ส่วนประกอบของปัญญาประดิษฐ์

รูปแบบการทำงานร่วมกันในแต่ละส่วนของปัญญาประดิษฐ์นั้นได้รับแรงบันดาลใจมาจาก “ความรู้สึกนึกคิดของมนุษย์” ส่วนที่เป็นหัวใจสำคัญคือ โครงข่ายประสาทเทียมหน้าที่หลักคือ “คิด” ให้ได้เช่นเดียวกับความคิดของมนุษย์ การทำงานของสมองมนุษย์นั้นน่าตื่นตาตื่นใจมาก เพราะสมองสามารถทำหน้าที่หลายๆ อย่างพร้อมกันให้กลายเป็นเรื่องง่าย ในสมองมนุษย์นั้นมีระบบประสาทนับพันล้านตัวที่ทำงานเชื่อมโยงกัน และมีจุดประสานกันที่เรียกว่า ไซแนป (Synapses) อีกล้านล้านจุดการที่ทุกจุดทำงานได้พร้อมกันเช่นนี้ ทำให้ระบบการทำงานของสมองมนุษย์นั้นยากที่จะเลียนแบบได้ และนั่นคือเป้าหมายที่เหล่านักวิทยาศาสตร์ นักคณิตศาสตร์ และผู้เชี่ยวชาญในแต่ละด้านพยายามที่จะพัฒนาระบบความนึกคิดของ โครงข่ายประสาทเทียมให้ทำงานได้เหมือนกับสมองมนุษย์

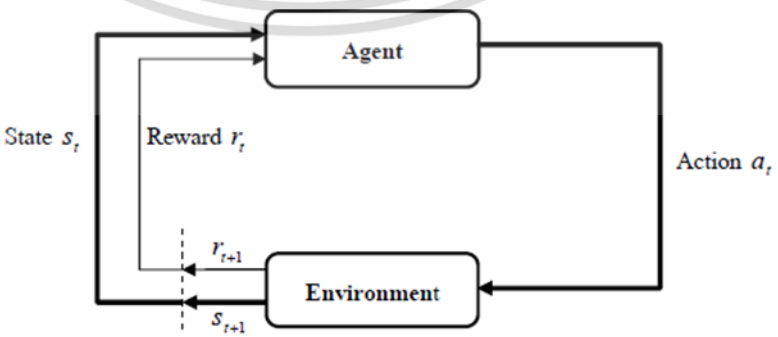
2.4 วิธีการเรียนรู้แบบเสริมกำลัง

วิธีการเรียนรู้แบบเสริมกำลัง (Reinforcement learning: RL) เป็นวิธีการหนึ่งของการทำการเรียนรู้ของเครื่อง [10] ดังรูปที่ 2.6



รูปที่ 2.6 ประเภทของการเรียนรู้ของเครื่อง [23]

วิธีการเรียนรู้แบบเสริมกำลังเป็นการเรียนรู้จากจุดมุ่งหมายโดยตรงแล้วจึงตัดสินใจในการทำตามจุดมุ่งหมายนั้น โดยโครงสร้างพื้นฐานในการเรียนรู้ด้วยการตัดสินใจจากสภาพแวดล้อมในลักษณะของ สถานะ (State) การกระทำ (Action) และผลรางวัล (Reward) เพื่อที่จะทำการเรียนรู้ในการเชื่อมโยงสถานะปัจจุบัน ไปสู่การเลือกการกระทำในลำดับถัดไป หรือเป็นการเรียนรู้เพื่อที่จะทำการตัดสินใจว่า หากสภาพแวดล้อมอยู่ในสถานะหนึ่งแล้วควรจะเลือกการกระทำใดต่อไปที่จะก่อให้เกิดผลของการกระทำที่ดี โดยมีวงรอบการทำงานดังรูปที่ 2.7 ซึ่งเริ่มจากเมื่อตัวกระทำการตัดสินใจของระบบ (Agent) ได้รับสถานะและผลรางวัลที่เกิดขึ้น ตัวกระทำการตัดสินใจจะทำการตัดสินใจที่จะเลือกกระทำการอย่างใดอย่างหนึ่ง จากนั้นตัวกระทำการตัดสินใจจะรับทราบถึงการเปลี่ยนไปของสถานะและผลรางวัลที่ได้รับกลับมา ระบบจะดำเนินไปและได้รับผลรางวัลกลับมาอยู่โดยตลอดจนกระทั่งตัวกระทำการตัดสินใจเกิดการเรียนรู้ว่าเมื่ออยู่ในสถานะใดจะต้องเลือกการกระทำใด จึงจะได้ผลรางวัลระยะยาวสูงสุด (Long-term expect reward)



รูปที่ 2.7 วงรอบการทำงานของวิธีการเรียนรู้แบบเสริมกำลัง [10]

โดยที่

Reward r_t คือ ผลรางวัลในระยะยาวที่ได้จากการเลือกการกระทำ ณ เวลา t

State s_t คือ สถานะของระบบ ณ เวลา t

Action a_t คือ การกระทำที่ตัวกระทำการตัดสินใจเลือกกระทำ ณ เวลา t

Reward r_{t+1} คือ ผลรางวัลที่ได้จากการกระทำในเวลา $t+1$

State s_{t+1} คือ สถานะของสิ่งแวดล้อมในเวลา $t+1$

เมื่อ s แทนสถานะของสิ่งแวดล้อม S เป็นเซตสถานะทั้งหมดที่เป็นไปได้ โดยที่ $s_t \in S$ และ $A(s_t)$ เป็นเซตของการกระทำทั้งหมดที่เป็นไปได้ในแต่ละสถานะของระบบ โดยที่ $a_t \in A(s_t)$ และ $r_{t+1} \in R$ เมื่อ t คือเวลาในแต่ละขั้นตอนโดย t = 0,1,2,3, ...

โดยทั่วไปแล้ววิธีการเรียนรู้แบบเสริมกำลังจะประกอบด้วย 4 องค์ประกอบหลัก [10] ดังนี้

1. กฎควบคุม (Policy) คือ ตัวกำหนดแนวทางของการเรียนรู้และการตัดสินใจ ในการเลือกการกระทำในลำดับถัดไป เพื่อให้ได้ผลรางวัลเฉลี่ยในระยะยาวที่สูงสุด ถ้าให้กฎควบคุม แทนด้วย π ที่ได้มาจากการหาค่าสูงสุดของผลเฉลี่ยรางวัลสะสมที่เรียกว่า ค่าคิว (Q-value) แทนด้วย $Q(s,a)$ เพื่อนำไปใช้เป็นฟังก์ชันในการเลือกการกระทำดังสมการที่ (2.1)

$$\pi(s) = arg\max_{a} Q(s,a) \tag{2.1}$$

2. ฟังก์ชันผลรางวัล (Reward function) คือ ฟังก์ชันที่ใช้คำนวณผลรางวัลในระยะยาวของระบบที่ได้ดำเนินอยู่ ณ ขณะนั้น โดยผลรางวัลนี้จะเกิดจากการตัดสินใจในการเลือก การกระทำเมื่อระบบอยู่ในสถานะต่างๆ โดยหากเป็นการตัดสินใจที่ถูกต้องจะได้ผลรางวัลในระดับที่สูง แต่หากเป็นการตัดสินใจที่ถูกต้องน้อยกว่าหรือเป็นการตัดสินใจที่ผิดจะได้รับผลรางวัลในระดับที่ต่ำลดหลั่นกันลงมา ซึ่งผลรางวัลนี้จะเป็นตัวแสดงถึงความสามารถของตัวตัดสินใจว่าสามารถทำการตัดสินใจได้ดีหรือไม่ในช่วงเวลา ณ ขณะนั้น จุดมุ่งหมายหลักของการแก้ปัญหาด้วยวิธีการเรียนรู้แบบเสริมกำลัง คือ ต้องการการเรียนรู้เพื่อให้ได้กฎควบคุมในการตัดสินใจที่ส่งผลให้ได้ผลรางวัลในระยะยาวที่สูงที่สุด ซึ่งจะแสดงถึงการตัดสินใจที่ดีของวิธีการเรียนรู้แบบเสริมกำลัง หากพิจารณาผลรางวัลที่เกิดขึ้น ณ เวลา t กำหนดเป็น $g(s_t, a_t)$

เมื่อผลรวมของผลรางวัลในระยะยาวแทนด้วย R_t จะได้ค่าผลรวมของผลรางวัลที่เกิดขึ้นตั้งแต่เวลา t จนถึงเวลา T ดังสมการที่ (2.2) ซึ่งวิธีการเรียนรู้แบบเสริมกำลัง มีเป้าหมายต้องการที่จะให้ได้ R_t ที่มากที่สุด

$$R_t = g(s_t, a_t) + g(s_{t+1}, a_{t+1}) + g(s_{t+2}, a_{t+2}) + \cdots + g(s_{T-1}, a_{T-1}) \tag{2.2}$$

เมื่อ R_t คือ ผลรวมของผลรางวัลของช่วงเวลาตั้งแต่เวลา t จนถึงเวลา T

3. แวลู่ฟังก์ชัน (Value function) คือ ฟังก์ชันที่ใช้คำนวณหาผลรางวัลในระยะยาวที่คาดว่าจะได้รับหากเลือกการกระทำนั้นๆ ภายใต้กฎควบคุมเดียวกัน ในการจำลองการกระทำซึ่งตัวกระทำการตัดสินใจ จะใช้ผลรางวัลนี้ในการตัดสินใจเลือกการกระทำในลำดับถัดไป โดยถ้าให้กฎควบคุม π เป็นกฎควบคุมที่ทำการเชื่อมโยงระหว่างสถานะ s โดยที่ $s \epsilon S$ และการกระทำ a โดยที่ $a \epsilon A$ และดำเนินไปภายใต้กฎการควบคุม π นี้ ดังสมการที่ (2.3) ซึ่งเป็นสมการสถานะแวลู่ฟังก์ชัน (State-value function) สำหรับกฎควบคุม π

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|s_t = s\} \tag{2.3}$$

เมื่อ $E_{\pi}\{\}$ เป็นค่าคาดหวังเมื่อตัวกระทำการตัดสินใจกระทำตามกฎควบคุม π และ γ เป็นอัตราการลดทอน (Discount rate) ที่มีค่าระหว่าง $0 \leq \gamma \leq 1$

4. แบบจำลองของสภาวะแวดล้อม (Model of environment) คือแบบจำลองของสิ่งแวดล้อมที่จะนำวิธีการเรียนรู้แบบเสริมกำลังเข้าไปประยุกต์ใช้ ซึ่งต้องสามารถที่จะแสดงพฤติกรรมได้เหมือนกับสภาพแวดล้อมที่จะนำมาประยุกต์ใช้ โดยให้ $P_{ss'}^{a}$ เป็นความน่าจะเป็นที่จะเกิดการเปลี่ยนสถานะจากการที่สภาวะแวดล้อมอยู่ที่สถานะ s จะเปลี่ยนสถานะเป็น s' เมื่อเลือกการกระทำ a ดังสมการที่ (2.4)

$$P_{ss'}^{a} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \tag{2.4}$$

การนำวิธีการเรียนรู้แบบเสริมกำลังมาใช้ในการแก้ปัญหานั้น จะเป็นการหากฎควบคุมที่จะก่อให้เกิดผลรางวัลสะสมระยะยาวสูงที่สุด โดยจะทำการปรับปรุงกฎควบคุมนี้ให้ดีขึ้นอยู่โดยตลอด ดังนั้นกฎควบคุมใหม่จึงดีกว่าหรือเท่ากับกฎควบคุมเดิม ซึ่งทำให้ได้กฎควบคุมอย่างน้อยหนึ่งกฎควบคุมที่ดีกว่าหรือเท่ากับกฎควบคุมอื่นๆ ซึ่งเรียกว่า กฎควบคุมที่เหมาะสม (Optimal policy) และเมื่อให้ระบบดำเนินไปตามกฎควบคุมนี้ จะได้สถานะแวลู่ฟังก์ชันที่เป็นสถานะแวลู่ฟังก์ชันที่เหมาะสม (Optimal state-value function) ซึ่งแทนด้วย V^* ดังสมการที่ (2.5)

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{2.5}$$

เมื่อทุกๆ $s \epsilon S$ ให้ระบบดำเนินไปตามกฎควบคุมที่เหมาะสมนี้ จะได้การกระทำแวลู่ฟังก์ชันที่เหมาะสม (Optimal action-value function) ซึ่งแทนด้วย Q^* ดังสมการที่ (2.6)

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{2.6}$$

เมื่อทุกๆ $s \in S$ และ $a \in A(s)$

สำหรับในทุกๆ คู่ของสถานะและการกระทำ (State-action pair) (s, a) จะได้ฟังก์ชันของค่าคาดหวังที่จะได้รับเมื่อเลือกการกระทำ a และสิ่งแวดล้อมอยู่ในสถานะ s และดำเนินไปด้วยกฎควบคุมที่เหมาะสม จะได้ความสัมพันธ์ของการกระทำแวลู่ฟังก์ชันที่เหมาะสม Q^* และสถานะแวลู่ฟังก์ชันที่เหมาะสม V^* ดังสมการที่ (2.7)

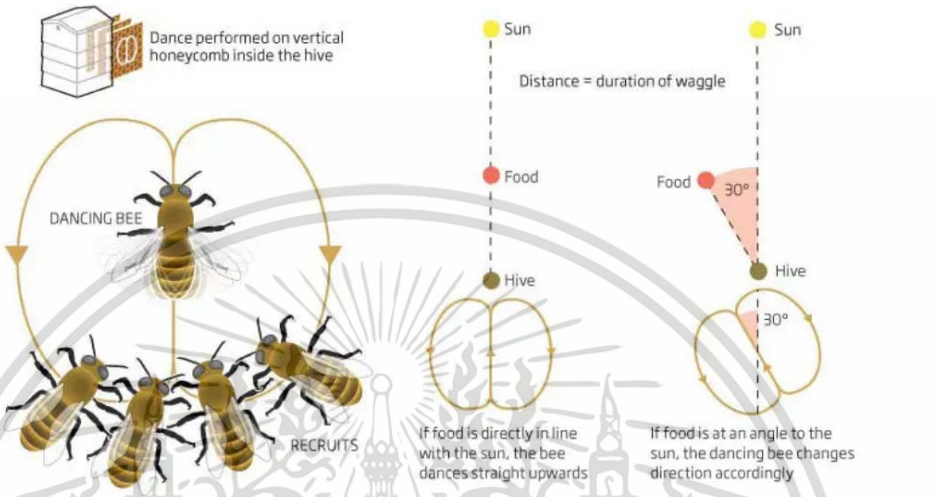
$$Q^*(s,a) = E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\} \quad (2.7)$$

2.5 ขั้นตอนวิธีอาณานิคมผึ้งเทียม

ขั้นตอนอาณานิคมผึ้ง (Bee colony algorithm) [24, 25] ได้ถูกนำเสนอโดย Pham และคณะ โดยวิธีขั้นตอนอาณานิคมผึ้งเป็นขบวนการในการหาค่าที่เหมาะสมที่สุด ที่เลียนแบบมาจากพฤติกรรมในการหาอาหารของฝูงผึ้ง และเป็นกระบวนการหาคำตอบชนิดหนึ่งที่อยู่ในกลุ่มที่มีแนวคิดมาจากการเลียนแบบพฤติกรรมของฝูงแมลง (Swarm based optimization algorithm) โดยลักษณะเด่นของกระบวนการหาคำตอบในกลุ่มนี้ จะใช้หลักการในการสุ่มค่าตัวแปรด้วยจำนวนประชากรจำนวนหนึ่งเข้าไป แล้วนำผลลัพธ์มาทำการคำนวณ การคัดเลือกและการปรับค่า แล้วทำการคำนวณวนรอบซ้ำจนกระทั่งได้คำตอบที่เหมาะสมในที่สุด ลักษณะของฝูงผึ้งในธรรมชาติ ผึ้งจะบินออกไปหาอาหารในระยะทางไกลมากกว่า 14 กิโลเมตรในหลายๆ ทิศทางพร้อมๆ กัน โดยมีเป้าหมายในการสำรวจหาแหล่งอาหารเพื่อให้ได้อาหารจำนวนมาก ฝูงผึ้งจะถูกส่งให้ไปสำรวจหาแหล่งอาหารที่มีอาหารอุดมสมบูรณ์ ตามทฤษฎีแหล่งที่มีดอกไม้ซึ่งให้ปริมาณน้ำหวานจากเกสรสูงและมีคุณภาพดี ก็จะมีผึ้งจำนวนมากกว่าที่บริเวณอื่น เช่นเดียวกันกับบริเวณที่มีแหล่งอาหารน้อยกว่าและคุณภาพต่ำกว่า ก็จะมีผึ้งจำนวนที่น้อยกว่าในบริเวณนั้น

ขบวนการในการหาอาหารของผึ้งเริ่มต้นจาก การส่งผึ้งสังเกตการณ์ไปสำรวจหาแหล่งดอกไม้ ผึ้งสังเกตการณ์จะบินสุ่มไปยังที่หนึ่งและสุ่มย้ายไปยังที่อื่นๆ ในช่วงฤดูเก็บเกี่ยวอาหาร ฝูงผึ้งจะสุ่มสำรวจหาแหล่งอาหารเมื่อผึ้งสังเกตการณ์บินกลับมาที่รัง ผึ้งตัวที่พบแหล่งอาหาร ที่มีคุณภาพสูงกว่ามาตรฐานเช่น มีส่วนประกอบของน้ำตาลสูงจะนำน้ำหวานมาเก็บ แล้วไปยังลานเต้น (Dance floor) เพื่อทำการเต้นที่เรียกว่า “เต้นแกว่ง” (Waggle dance) การเต้นนั้นมีความสำคัญสำหรับการสื่อสารของฝูงผึ้งเป็นอย่างมาก เนื่องจากผึ้งจะใช้ช่วงระหว่างการเต้นสื่อสารข้อมูลสามชุดที่เกี่ยวกับแหล่งของดอกไม้นั่นคือ ทิศทางของดอกไม้ ระยะทางที่ห่างจากรัง และระดับคุณภาพของน้ำหวาน ซึ่งข้อมูลเหล่านี้จะทำให้ผึ้งตัวอื่นบินออกไปหาแหล่งดอกไม้ได้อย่างแม่นยำโดยไม่ต้องมีแผนที่ ฝูงผึ้งจะนำข้อมูลที่ได้จากการสำรวจภายนอกแต่ละชุดมาสื่อสารระหว่างกันในช่วงที่ทำการเต้นแกว่ง และทำการเปรียบเทียบทั้งในด้านของคุณภาพของอาหาร และพลังงานที่จะต้องใช้ในการออกไปหาอาหาร หลังจากการเต้นแกว่งในลานเต้น ผึ้งสังเกตการณ์ก็จะกลับไปยังแหล่งดอกไม้ร่วมกับผึ้งตัวอื่นๆ ซึ่งอยู่

ในรัง ผึ้งจำนวนมากกว่าจะถูกส่งไปยังบริเวณที่เชื่อว่าจะมีอาหารที่มีคุณภาพสูงกว่า และเช่นเดียวกัน ผึ้งจำนวนน้อยกว่าก็จะถูกส่งไปยังพื้นที่ที่เชื่อว่าจะมีอาหารคุณภาพต่ำกว่าซึ่งจากวิธีการนี้จะทำให้ฝูงผึ้งหาอาหารได้อย่างรวดเร็วและมีประสิทธิภาพ ซึ่งการเต้นแกว่งเพื่อสื่อสารข้อมูลของแหล่งอาหารของผึ้งแสดงได้ดังรูปที่ 2.8



รูปที่ 2.8 การเต้นแกว่งของผึ้งเพื่อสื่อสารข้อมูลของแหล่งอาหาร [26]

ขั้นตอนวิธีอาณานิคมผึ้งเทียม (Artificial bee colony algorithm: ABC algorithm) [6, 8] เป็นขั้นตอนวิธีที่ใช้ในการแก้ปัญหาการหาค่าหรือวิธีการที่เหมาะสมที่สุด จากค่าหรือวิธีการที่เป็นไปได้ทั้งหมดเพื่อให้ได้คำตอบที่ต้องการ เช่น ค่าน้อยที่สุด หรือมากที่สุดของปัญหาที่ต้องการแก้ ขั้นตอนวิธีอาณานิคมผึ้งเทียมเป็นความฉลาดแบบกลุ่ม (Swarm intelligent: SI) ที่ได้แรงบันดาลใจมาจากธรรมชาติและเป็นอัลกอริทึมที่เป็นที่นิยมนำใช้ในการแก้ปัญหาเพื่อหาค่าคำตอบที่ดีที่สุด ขั้นตอนวิธีอาณานิคมผึ้งเทียมได้เลียนแบบพฤติกรรมการออกหาอาหารของอาณานิคมผึ้ง ประชากรของอาณานิคมผึ้งเทียม แบ่งออกเป็น 3 กลุ่ม คือ ผึ้งงาน (Employed bees) ผึ้งเฝ้าดู (Onlooker bees) และผึ้งสำรวจ (Scout bees) โดยผึ้งงานจะออกหาอาหารแล้วกลับมาให้ข้อมูลเกี่ยวกับแหล่งอาหารที่พบให้กับผึ้งเฝ้าดูที่รออยู่ในรัง ผึ้งเฝ้าดูจะเลือกผึ้งงานเพื่อตามออกไปเก็บอาหารขึ้นอยู่กับข้อมูลที่ได้รับจากผึ้งงาน ถ้าอาหารแหล่งใดถูกเก็บจนหมด ผึ้งงานที่เป็นเจ้าของแหล่งอาหารนั้นจะกลายไปเป็นผึ้งสำรวจ เพื่อสุ่มสำรวจหาแหล่งอาหารใหม่ ขั้นตอนวิธีอาณานิคมผึ้งเทียมสำหรับปัญหาการหาค่าพารามิเตอร์ที่เหมาะสมที่สุดมีขั้นตอนหลักดังนี้

1) เริ่มต้นสร้างเวกเตอร์พารามิเตอร์ให้กับผึ้งงาน
2) ผึ้งงานสร้างเวกเตอร์พารามิเตอร์ใหม่จากเวกเตอร์พารามิเตอร์เดิมแล้วเลือกเวกเตอร์พารามิเตอร์ที่ให้ค่าความเหมาะสมมากกว่า

3) ผึ้งเฝ้าดูแต่ละตัวเลือกผึ้งงานโดยขึ้นอยู่กับค่าความเหมาะสมของผึ้งงาน แล้วสร้างเวกเตอร์พารามิเตอร์ใหม่จากเวกเตอร์พารามิเตอร์ของผึ้งงานที่เลือก ซึ่งผึ้งงานจะเลือกเวกเตอร์พารามิเตอร์ที่ให้ค่าความเหมาะสมมากกว่า
4) จดจำเวกเตอร์พารามิเตอร์ที่ดีที่สุดตั้งแต่เริ่มต้นจนถึงขั้นตอนนี้
5) หาเวกเตอร์พารามิเตอร์ของผึ้งงานที่ต้องละทิ้งแล้วกำหนดให้เป็นผึ้งสำรวจที่สร้างเวกเตอร์พารามิเตอร์ขึ้นมาใหม่
6) ทำซ้ำขั้นตอนที่ 2-5 จนกระทั่งครบตามจำนวนรอบที่กำหนดไว้

ขั้นตอนที่ 1 เวกเตอร์พารามิเตอร์หรือตำแหน่งแหล่งอาหารของผึ้งงาน $\bar{x}_i$ ประกอบด้วยสมาชิก x_{ij} เมื่อ $i = 1,2,\dots,SN$ โดยที่ SN คือ จำนวนของผึ้งงานทั้งหมดภายในอาณานิมผึ้งเทียม และ $j = 1,2,\dots,D$ โดยที่ D จำนวนพารามิเตอร์ทั้งหมดที่ต้องการหาค่าที่เหมาะสมโดย x_{ij} สร้างโดยสมการที่ (2.8) แล้วคำนวณค่าฟังก์ชันวัตถุประสงค์และค่าความเหมาะสมโดยสมการที่ (2.9)

$$x_{ij} = l_j + rand(0,1) * (u_j - l_j) \tag{2.8}$$

เมื่อ l_j และ u_j คือ ค่าต่ำสุดและสูงสุดของพารามิเตอร์ x_{ij} ตามลำดับ

$rand$ คือ ค่าสุ่มที่มีค่าระหว่าง 0 ถึง 1

$$fit_i(\overrightarrow{x_i}) = \begin{cases} \dfrac{1}{1 + f_i(\overrightarrow{x_i})} & , \quad f_i(\overrightarrow{x_i}) \geq 0 \\ 1 + |f_i(\overrightarrow{x_i})| & , \quad f_i(\overrightarrow{x_i}) < 0 \end{cases} \tag{2.9}$$

เมื่อ $fit_i(\overrightarrow{x_i})$ คือ ค่าของฟังก์ชันวัตถุประสงค์สำหรับเวกเตอร์พารามิเตอร์ $\overrightarrow{x_i}$ ซึ่งเป็นฟังก์ชันที่ต้องการหาค่าพารามิเตอร์ที่เหมาะสม

ขั้นตอนที่ 2 ผึ้งงานสร้างเวกเตอร์พารามิเตอร์ใหม่ $\overrightarrow{v_i}$ จาก $\overrightarrow{x_i}$ โดยหาได้จากสมการที่ (2.10) แล้วคำนวณค่าฟังก์ชันวัตถุประสงค์ และค่าความเหมาะสมเช่นเดียวกันกับขั้นตอนที่ 1

$$v_{ij} = x_{ij} + \varphi \times (x_{ij} - x_{rj}) \tag{2.10}$$

เมื่อ $\overrightarrow{x_r}$ คือ เวกเตอร์พารามิเตอร์ที่สุ่มจาก $\vec{x}$ ที่ไม่ใช่เวกเตอร์พารามิเตอร์เดียวกันกับ $\overrightarrow{x_i}$

j คือ จำนวนเต็มสุ่มที่มีค่าระหว่าง 1 ถึง D และ φ เป็นตัวเลขสุ่มที่มีค่าระหว่าง -1 ถึง 1 ผึ้งงานจะเลือกเวกเตอร์พารามิเตอร์ระหว่าง $\overrightarrow{x_i}$ และ $\overrightarrow{v_i}$ ที่ให้ค่าความเหมาะสมที่มากกว่า

ขั้นตอนที่ 3 ผึ้งเฝ้าดูเลือกผึ้งงานโดยพิจารณาค่าความน่าจะเป็น p_i ที่คำนวณจากค่า $fit_i(\overrightarrow{x_i})$ โดยคำนวณจากสมการที่ (2.11)

$$p_i = \frac{fit_i(\vec{x_i})}{\sum_{i=1}^{SN} fit_i(\vec{x_i})} \quad (2.11)$$

กระบวนการเลือกผึ้งงานของผึ้งเฝ้าดู จะเริ่มจากผึ้งเฝ้าดูแต่ละตัวสุ่มค่าระหว่าง 0 ถึง 1 มาหนึ่งค่าแล้วเทียบกับ ค่า p_i ถ้าค่าสุ่มนั้นมีค่าน้อยกว่า ผึ้งเฝ้าดูก็จะเลือกผึ้งงานตัวนั้น ผึ้งเฝ้าดูจะสร้างเวกเตอร์พารามิเตอร์ใหม่จากเวกเตอร์พารามิเตอร์ของผึ้งงานที่เลือก พร้อมทั้งคำนวณค่าฟังก์ชันวัตถุประสงค์และค่าความเหมาะสม จากนั้นผึ้งงานจะเลือกเวกเตอร์พารามิเตอร์ที่ให้ค่าความเหมาะสมที่มากกว่าเช่นเดียวกันกับขั้นตอนที่ 2

ขั้นตอนที่ 4 ผึ้งงานตัวใดที่ไม่ถูกปรับค่าเวกเตอร์พารามิเตอร์ติดต่อกันจนถึงจำนวนที่กำหนดไว้ (Limit or abandonment criterion) ก็จะถูกกำหนดให้เป็นผึ้งสำรวจแล้วกำหนดเวกเตอร์พารามิเตอร์ใหม่ด้วยสมการที่ (2.8)

2.6 การหาค่าความเหมาะสมที่สุดด้วยวิธีอาณานิคมมด

วิธีอาณานิคมมด (Ant Colony Optimization: ACO) [27, 28] เป็นหนึ่งในกลุ่มอัลกอริทึมแบบเมตาฮิวริวติก อัลกอริทึมนี้มีการค้นหาเส้นทางที่เหมาะสมและเป้าหมายในการค้นหาคำตอบ โดยอาศัยการเลียนแบบพฤติกรรมตามธรรมชาติการหาอาหารของมด ซึ่งวิธีการของอาณาจักรมดที่ใช้ในการค้นหาคำตอบจะกำหนดจำนวนคำตอบเพียงหนึ่งคำตอบ ซึ่งวิธีการอาณาจักรมดได้ถูกนำเสนอโดย มาร์โค โดริโก (Macro Dorigo) ในปี 1992 โดยวิธีการระบบมดหรืออาณาจักรมดได้แรงบันดาลใจจากการออกหาอาหารของมด ซึ่งมดจะเดินทางจากรังสู่แหล่งอาหารและกลับมาที่รังหลังจากได้อาหารอีกครั้ง ในระหว่างการเดินทางไปหาอาหาร มดแต่ละตัวจะปล่อยสารเคมีที่เรียกว่า สารฟีโรโมน (Pheromone) เพื่อให้มดตัวอื่นๆ ได้เดินตามกลิ่นหรือเดินตามรอยของสารฟีโรโมน ซึ่งสารฟีโรโมนจะระเหยเมื่อเวลาผ่านไป การระเหยของสารฟีโรโมนจะมีอยู่ตลอดเวลาตามคุณสมบัติทางเคมี ดังนั้นหากระยะทางที่ยาวจนเกินไปจะทำให้สารฟีโรโมนระเหยหมดในระหว่างทาง หรือเส้นทางใดที่มดไม่มีการเลือกเดินในเส้นทางนั้น ปริมาณของสารฟีโรโมนก็จะน้อยลง จึงมีความเป็นไปได้ที่มดจะเลือกเส้นทางนั้นน้อยลง จนในที่สุดปริมาณสารฟีโรโมนบนเส้นทางนั้นจะถูกระเหยไปจนหมด ซึ่งเรียกกลไกนี้ว่า กระบวนการป้อนกลับแบบลบ (Negative feedback) หากระยะทางสั้นก็จะเกิดการระเหยไปหมด และหากมีการเดินทางของมดหลายตัวหรือมีการเดินทางของมดหลายรอบ ทำให้ค่าความเข้มข้นของสารฟีโรโมนนั้นมีค่าเพิ่มมากขึ้นซึ่งสามารถเรียกกลไกนี้ว่า กระบวนการป้อนกลับแบบบวก (Positive feedback) ในการเลือกเส้นทางในการเดินหาอาหารของมด มดทุกตัวจำเป็นต้องทำการสุ่มเลือกเส้นทางใดเส้นทางหนึ่งในการเดินทางหาอาหาร ดังนั้นแต่ละเส้นทางจึงมีโอกาสเท่ากันที่มดแต่ละตัวในฝูงจะเลือกเส้นทางนั้น เพื่อเดินไปยังแหล่งอาหาร กฎต่างๆ ที่ใช้ในวิธีอาณานิคมมดสามารถอธิบายได้ดังนี้

1) กฎการเปลี่ยนสถานะ (State transition rule) ของวิธีอาณานิคมมด

มดจะทำการเดินจากจุดหนึ่งไปยังอีกจุดหนึ่งหรือเรียกว่า การเปลี่ยนสถานะ มดจะทำการเลือกจุดที่จะไปโดยอาศัยกฎการเปลี่ยนสถานะที่มี 2 ทางเลือกที่เป็นไปได้ คือ การเลือกสำรวจเพื่อหาผลเฉลยที่เป็นไปได้ใหม่ (Exploration) หรือการเลือกผลเฉลยโดยอาศัยความรู้เดิมที่มีอยู่ก่อนแล้ว (Exploitation) โดยกำหนดให้จุดที่มดอยู่แทนด้วย i และจุดที่มดจะเดินทางไปแทนด้วย j และทำการเลือกจุดโดยใช้สมการที่ (2.12)

$$p_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}(t)]^{\beta}}{\sum_{k \epsilon allowed_k}[\tau_{ik}(t)]^{\alpha}[\eta_{ik}(t)]^{\beta}} & if\ j \epsilon\ allowed_k \\ 0 & otherwise \end{cases} \quad (2.12)$$

โดยที่ $allowed_k$ คือ เซตของจุดที่เหลือสำหรับมดตัวที่ k ในขณะที่มดอยู่ที่เมือง i

$\tau_{ij}(t)$ คือ ค่าฟีโรโมน ณ เวลา t

$\eta_{ij}(t)$ คือ ค่าข้อมูลฮิวริสติก (Heuristic information) หรือค่าความใกล้เคียงระยะทางจากเมือง i ไป j ณ เวลา t โดยกำหนดให้ $\eta_{ij}(t) = 1/d_{ij}$ และ d_{ij} คือระยะทางระหว่างเมือง i และ j

α, β คือ ค่าคงที่ระบุความสัมพันธ์ระหว่างค่าฟีโรโมนและค่าข้อมูลฮิวริสติก

2) กฎปรับฟีโรโมนวงกว้าง (Global updating rule)

เมื่อมดทุกตัวเดินทางครบรอบของเส้นทางโดยเดินผ่านทุกจุดแล้ว เส้นทางที่มีระยะในการเดินทางที่สั้นที่สุดจะมีการปล่อยฟีโรโมนซ้ำอีก เพื่อเป็นการสร้างเส้นทางนั้นให้ชัดเจนยิ่งขึ้น ซึ่งมดตัวที่ให้คำตอบที่ดีที่สุดเท่านั้นที่สามารถปรับเปลี่ยนค่าฟีโรโมน โดยสมการการเพิ่มฟีโรโมนแสดงดังสมการที่ (2.13)

$$\tau_{ij} \leftarrow (1-\alpha) \cdot \tau_{ij} + \alpha \cdot \Delta\tau_{ij} \quad (2.13)$$

เมื่อ α คือ อัตราการระเหยของฟีโรโมน ($0 \leq \alpha \leq 1$) ยิ่งค่าของ α มีค่ามากเท่าใด ผลกระทบของวิธีการแก้ปัญหาในอดีตก็จะยิ่งน้อยลงเท่านั้น

$\Delta\tau_{ij}$ สามารถหาได้จากสมการที่ (2.14)

$$\Delta\tau_{ij} = \begin{cases} (L_{gb})^{-1} & \text{ถ้า } (i,j) \in global - best - tour \\ 0 & \text{กรณีอื่นๆ} \end{cases} \quad (2.14)$$

เมื่อ L_{gb} คือ ระยะทางของเส้นทางที่สั้นที่สุดจากระยะทางทั้งหมด (Globally best tour) จากสมการที่ (2.13) จะเห็นว่า ฟีโรโมนที่เหลือในเส้นทางที่สั้นที่สุด เมื่อรวมกับอัตราการระเหยของฟีโรโมนที่ระเหยไปในเส้นทางที่สั้นที่สุด จะมีค่ากับฟีโรโมนที่มดปล่อยออกมาบนเส้นทางนั้น แต่จะแตกต่างกัน การปรับค่าฟีโรโมนวงแคบคือ ฟีโรโมนจะถูกปล่อยให้ระเหยไปบนเส้นทางที่มีระยะทางสั้นที่สุด และจะถูกปล่อยออกมาจากมดตัวถัดไปตามระยะทาง L_{gb} ดังสมการที่ (2.14) ซึ่งเป็นเส้นทางที่ดีที่สุดจากจำนวนเส้นทางทั้งหมด

2.7 ขั้นตอนวิธีการหาคำตอบเหมาะสมที่สุดแบบกลุ่มอนุภาค

ขั้นตอนวิธีการหาคำตอบเหมาะสมที่สุดแบบกลุ่มอนุภาค (Particle Swarm Optimization: PSO) [29-32] เป็นหนึ่งในขั้นตอนวิธีที่เป็นที่นิยมในการนำมาแก้ไขปัญหาการหาคำตอบที่เหมาะสมที่สุด ซึ่งเป็นแขนงหนึ่งของความฉลาดแบบกลุ่ม PSO ถูกพัฒนาขึ้นในปี ค.ศ.1995 โดยเจมส์ เคนเนดี (James Kennedy) และรัสเซล อีเบอฮาร์ท (Russell C. Eberhart) มีแนวคิดมาจากการเลียนแบบพฤติกรรมการหาอาหารของฝูงสัตว์ที่หาอาหารแบบเป็นฝูง โดยเฉพาะฝูงนกและฝูงปลา และทฤษฎีการเคลื่อนที่ (Velocity theory) ซึ่งต้องอาศัยการเคลื่อนที่กันเป็นกลุ่มและการเว้นระยะห่างระหว่างนก โดยเรียกนกแต่ละตัวว่าอนุภาค (Particles) แต่ละอนุภาคถูกแทนด้วยความเร็วและตำแหน่ง ทั้งนี้วิธีการหาคำตอบที่เหมาะสมที่สุดแบบกลุ่มอนุภาคเริ่มต้นทำงานด้วยการสุ่มอนุภาคลงบนพื้นที่การค้นหา (Search space) เพื่อสร้างประชากรเริ่มต้น จากนั้นทำการเคลื่อนที่บนพื้นที่การค้นหาตามค่าตำแหน่งที่ดีที่สุดของกลุ่ม (Global best: gbest) และตำแหน่งที่ดีที่สุดของอนุภาค (Personal best: pbest) โดยกระบวนการของวิธีการหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาคจะทำการปรับปรุงและค้นหาคำตอบไปเรื่อย ๆ จนกว่าจะพบค่าคำตอบที่เหมาะสมที่สุด หรือครบรอบที่กำหนดไว้ ค่าตำแหน่งที่ดีที่สุดของเพื่อนบ้านที่ได้ในรอบสุดท้ายจะถูกนำมาใช้เป็นคำตอบของวิธีการหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค

การทำงานของขั้นตอนวิธีการหาคำตอบเหมาะสมที่สุดแบบกลุ่มอนุภาค สามารถอธิบายได้ดังนี้

1.กำหนดค่าเริ่มต้นให้อนุภาค (Initial particles) โดยการสุ่มตำแหน่งและกำหนดความเร็วเริ่มต้นของอนุภาคทั้งหมดในพื้นที่การค้นหา

2.ประเมินค่าคำตอบของแต่ละอนุภาค (Evaluate) โดยประเมินจากค่าความเหมาะสม (Fitness value) ผ่านฟังก์ชันวัตถุประสงค์ (Objective function) เพื่อนำไปใช้ในการหาค่าตำแหน่งที่ดีที่สุดของอนุภาค (pbest) และตำแหน่งที่ดีที่สุดของกลุ่มอนุภาค (gbest)

3.การปรับปรุงค่า pbest (Update pbest) คือการที่แต่ละอนุภาคทำการเปรียบเทียบค่าความเหมาะสมของอนุภาคกับค่า pbest เดิมของตัวเอง และทำการปรับปรุงค่า pbest ของตัวเองด้วยค่าความเหมาะสมของอนุภาคที่มีค่าดีกว่า

4.การปรับปรุงค่า gbest (Update gbest) คือการที่แต่ละอนุภาคทำการเปรียบเทียบค่าความเหมาะสมของตนเองกับค่า gbest ของกลุ่ม และทำการปรับปรุงค่า gbest ของตัวเองด้วยค่าความเหมาะสมของอนุภาคที่มีค่าดีกว่า

5.การปรับปรุงความเร็ว (Update velocity) คือการปรับปรุงค่าความเร็วของแต่ละอนุภาค ที่ใช้ในการเคลื่อนที่ของอนุภาค โดยการนำค่า gbest pbest และค่าความเร็วเดิมมาใช้ในการคำนวณ โดยใช้สมการที่ (2.15)

$$\vec{v}_k^{t+1} = \omega\vec{v}_k^t + c_1 r_1(pbest - \vec{x}_k^t) + c_2 r_2(gbest - \vec{x}_k^t) \quad (2.15)$$

โดยที่

$\vec{v}_k^{t+1}$ คือ เวกเตอร์ความเร็วของอนุภาค

$\vec{v}_k^t$ คือ เวกเตอร์ความเร็วเดิมของอนุภาค

$pbest$ คือ ตำแหน่งที่ดีที่สุดของอนุภาค

$gbest$ คือ ตำแหน่งที่ดีที่สุดของกลุ่ม

c_1, c_2 คือ สัมประสิทธิ์การเร่งความเร็วไปยังตำแหน่งที่ดีที่สุดของอนุภาคและของกลุ่ม

ω คือ ค่าการถ่วงน้ำหนัก (Inertia weight)

r_1, r_2 คือ ค่าที่ได้จากการสุ่มในช่วง [0,1]

$\vec{x}_k^t$ คือ ตำแหน่งปัจจุบันของอนุภาค

6. การปรับปรุงตำแหน่ง (Update position) คือการปรับปรุงตำแหน่งของอนุภาคให้เคลื่อนที่ไปยังตำแหน่งใหม่ โดยนำค่าความเร็วที่ได้จากการปรับปรุงในขั้นตอนก่อนหน้ามาใช้ในการคำนวณ โดยสามารถคำนวณได้จากสมการที่ (2.16)

$$\vec{x}_k^{t+1} = \vec{x}_k^t + \vec{v}_k^{t+1} \quad (2.16)$$

โดยที่

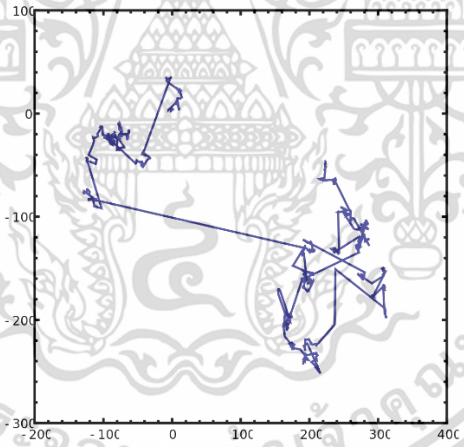
$\vec{x}_k^{t+1}$ คือ เวกเตอร์ตำแหน่งใหม่ของอนุภาค

$\vec{v}_k^{t+1}$ คือ เวกเตอร์ความเร็วในรอบปัจจุบันของอนุภาค

7.ตรวจสอบค่าคำตอบหรือค่าความเหมาะสม (Condition satisfied) และกำหนดเงื่อนไขในการหยุดค้นหาคำตอบ

2.8 ขั้นตอนวิธีการหาค่าเหมาะสมที่สุดแบบนกกาเหว่า

การค้นหาแบบนกกาเหว่า (Cuckoo Search: CS) [33] เป็นหนึ่งในวิธีเมตาฮิวริสติก ที่ถูกนำเสนอในปี ค.ศ.2009 โดยหยาง (X. S. Yang) และเด็บ (S. Deb) เพื่อนำมาใช้ในการแก้ปัญหาการค้นหาค่าที่เหมาะสมที่สุด โดยมีแรงบันดาลใจการนำพฤติกรรมการฝากเลี้ยง (Brood parasitism) ของนกกาเหว่า เนื่องจากนกกาเหว่าไม่สร้างรังและไม่ฟักไข่เอง แต่จะไปวางไข่ในรังของนกอื่นแล้วกกไข่และฟักไข่ให้ เรียกว่า “นกเจ้าของรัง (Host bird)” แต่ถ้านกเจ้าของรังรู้ว่าไม่ใช่ของมัน มันอาจจะโยนไข่ของนกกาเหว่าทิ้ง หรือทิ้งรังแล้วสร้างรังใหม่ที่อื่น โดยทั่วไปไข่ของนกกาเหว่าจะฟักเป็นตัวเร็วกว่าไข่ของนกเจ้าของรัง จากการศึกษาต่างๆ พบว่า พฤติกรรมการบินของสัตว์หลายชนิดมีระยะการเคลื่อนที่ไม่สม่ำเสมอและไร้รูปแบบ ซึ่งแสดงให้เห็นถึงลักษณะการเคลื่อนที่แบบเลวี (Lévy Flights) ดังรูปที่ 2.9 หรือแม้กระทั่งแสงก็สามารถเกี่ยวข้องกับลักษณะการเคลื่อนที่แบบ Lévy Flights ได้เช่นกัน ดังนั้นการเคลื่อนที่แบบ Lévy Flights จึงนิยมนำมาใช้ในการแก้ปัญหาการค้นหาค่าที่เหมาะสมที่สุด [34, 35] ผลจากการศึกษาวิจัยที่ผ่านมาจนถึงในปัจจุบันส่วนใหญ่พบว่า วิธีการหาค่าเหมาะสมที่สุดแบบนกกาเหว่า มีประสิทธิภาพสูงกว่าขั้นตอนวิธีการหาคำตอบเหมาะสมที่สุดแบบกลุ่มอนุภาค (PSO) และขั้นตอนวิธีเชิงพันธุกรรม (Genetic Algorithm: GA)



รูปที่ 2.9 ตัวอย่างลักษณะการเคลื่อนที่แบบ Lévy Flights [34]

ขั้นตอนวิธีการหาค่าเหมาะสมที่สุดแบบนกกาเหว่า ประกอบด้วยสมาชิกในรังหรือไข่ ซึ่งไข่แต่ละฟองใช้แทนคำตอบของปัญหา ไข่ของนกกาเหว่าที่มีคุณภาพ (มีค่าใกล้เคียงกับคำตอบที่ดีที่สุด) จะมีลักษณะคล้ายคลึงกับไข่ของนกเจ้าของรัง นั้นหมายความว่า ไข่ของนกกาเหว่ามีโอกาสที่จะรอดพ้นจากการถูกค้นพบและโยนทิ้งไป และเติบโตเป็นนกกาเหว่ารุ่นถัดไป ส่วนไข่ของนกกาเหว่าที่ไม่มีคุณภาพ (มีค่าห่างไกลคำตอบที่ดีที่สุด) จะมีค่าความน่าจะเป็นในการที่ไข่ของนกกาเหว่าจะถูกค้นพบจากนกเจ้าของรังอยู่ระหว่าง 0 ถึง 1 ($p_a \in [0,1]$) และอาจจะถูกโยนทิ้ง หรือนกเจ้าของรังจะทิ้งรัง

หรือสร้างรังใหม่ในตำแหน่งอื่น การกระจายประชากรเริ่มต้นของรัง (x_i^t) จะถูกสร้างขึ้นโดยการสุ่ม หลังจากนั้นจะใช้วิธีการค้นหาแบบนกกาเหว่าที่มีลักษณะการเคลื่อนที่แบบ Lévy Flights สุ่มเลือกตำแหน่งรังถัดไป (x_i^{t+1}) จนกระทั่งได้ประชากรหรือไข่ที่เป็นคำตอบแทนได้ด้วยสมการที่ (2.17)

$$x_i^{t+1} = x_i^t + \propto s \otimes H(p_a - \varepsilon) \otimes (x_j^t - x_k^t) \tag{2.17}$$

โดยที่

x_j^t และ x_k^t คือ การแก้ปัญหาที่แตกต่างกันสองแบบที่เลือกโดยการสุ่มเปลี่ยน

$H(u)$ คือ ฟังก์ชัน Heaviside

ε คือ ตัวเลขสุ่มที่ถูกสุ่มจากการแจกแจงแบบเดียวกัน

s คือ ระยะกระโดด (Step size)

$\otimes$ คือ ผลคูณของเวกเตอร์

ในอีกทางหนึ่งการเดินสุ่มทั่วพื้นที่นั้นดำเนินการโดยใช้ Lévy Flights โดยใช้สมการที่ (2.18) - (2.20)

$$x_i^{t+1} = x_i^t + \propto \oplus \mathrm{L\acute{e}vy}(\lambda) \tag{2.18}$$

$$\mathrm{L\acute{e}vy} \approx a = t^{-\lambda},\ 1 < \lambda \leq 3 \tag{2.19}$$

$$\mathrm{L\acute{e}vy}(\lambda) = \left[\frac{\Gamma(1+\lambda)\cdot\sin\left(\frac{\pi\lambda}{2}\right)}{\Gamma\left(\frac{1+\lambda}{2}\right)\cdot\lambda\cdot 2^{\left(\frac{\lambda-1}{2}\right)}}\right]^{\frac{1}{\lambda}} \tag{2.20}$$

โดยที่ x_i^{t+1} คือ ตำแหน่งของรังในรอบใหม่

x_i^t คือ ตำแหน่งของรังในรอบเดิม

$\propto$ คือ ขนาดของการค้นหา (Step size) ซึ่งค่าพารามิเตอร์ที่มีค่ามากกว่า 0 $(\propto > 0)$

เมื่อ λ คือ ขนาดขั้นตอนที่เกี่ยวข้องกับมาตราส่วนของปัญหาที่สนใจ การปรับปรุงสมการที่ (2.18) จะสามารถพิจารณาสมการสโตแคสติก (Stochastic equation) สำหรับการเดินแบบสุ่ม (Random walk) [36] โดยทั่วไปการเดินแบบสุ่มเป็นห่วงโซ่มาร์คอฟ (Markov chains) ซึ่งสถานะหรือสถานที่ ต่อไปขึ้นอยู่กับตำแหน่งปัจจุบันเท่านั้น (เทอมแรกในสมการก่อนหน้า) และความน่าจะเป็นในการเปลี่ยนแปลง (เทอมที่สอง) อย่างไรก็ตามโซลูชันใหม่ควรมีการสร้างโดยการสุ่มฟิลด์ในพื้นที่ห่างไกล และสถานที่ควรจะไกลพอจากคำตอบที่ดีที่สุดในปัจจุบัน สิ่งนี้จะทำให้แน่ใจว่าระบบจะไม่ติดอยู่ในค่า ดีที่สุดเฉพาะที่ (Local optimum)

ขั้นตอนวิธีการหาค่าเหมาะสมที่สุดแบบนกกาเหว่า ตั้งอยู่บนแนวคิดพื้นฐานของวิธีการค้นหาแบบนกกาเหว่า 3 ข้อดังต่อไปนี้

1. นกกาเหว่าแต่ละตัวจะวางไข่จำนวน 1 ฟองต่อครั้งในรังที่มีการสุ่มเลือก
2. รังที่ดีและไข่ที่มีคุณภาพสูง (คำตอบของปัญหา) จะดำเนินการไปยังรุ่นถัดไปได้
3. กำหนดจำนวนรังของนกเจ้าของรังที่ใช้ได้ (n) และไข่ที่วางโดยนกกาเหว่าจะถูกค้นพบโดยนกเจ้าของรัง ซึ่งมีค่าความน่าจะเป็นอยู่ระหว่าง 0 ถึง 1 $(p_a \in [0,1])$ ในกรณีนี้ นกเจ้าของรังสามารถกำจัดไข่ หรือทิ้งรังไปและสร้างรังใหม่

จากแนวคิดพื้นฐานทั้ง 3 ข้อ สามารถนำมาสรุปเป็นขั้นตอนพื้นฐานของขั้นตอนวิธีการหาค่าเหมาะสมที่สุดแบบนกกาเหว่า ได้ดังนี้

1. กำหนดฟังก์ชันวัตถุประสงค์ $f(x), x = (x_1, x_2, \dots, x_d)^T$
2. สร้างประชากรเริ่มต้นของ n โดยเป็นจำนวนรังเริ่มต้น x_i เมื่อ $i = 1,2, \dots, n$
3. ตรวจสอบว่าครบรอบตามเกณฑ์ที่กำหนดหรือไม่ (ถึงเกณฑ์ที่ต้องหยุด) ถ้าใช่ให้ทำตามขั้นตอนที่ 9 ถ้าไม่ให้ทำตามขั้นตอนที่ 4
4. ประเมินคุณภาพของรังจากฟังก์ชันวัตถุประสงค์ F_i หลังจากนั้นเลือกรังใหม่มาประเมินคุณภาพตามฟังก์ชัน F_j
5. นำค่าที่ได้มาเปรียบเทียบกัน ถ้าหากค่า F_i มากกว่าค่า F_j ให้แทนที่รัง j เป็นคำตอบใหม่
6. โดยที่การค้นพบรังใหม่นั้นต้องมีค่าความน่าจะเป็นมากกว่าค่า p_a ที่กำหนดไว้ สำหรับรังที่ไม่ผ่านค่า p_a ให้ทิ้งค่ารังแล้วสุ่มเลือกค่ารังใหม่
7. เก็บคำตอบที่ดีที่สุดหรือรังที่มีคุณภาพ
8. เรียงลำดับผลลัพธ์ที่ได้ จากนั้นกลับไปทำขั้นตอนที่ 3
9. ผลลัพธ์สุดท้ายจะได้คำตอบที่ดีที่สุด หรือรังที่มีคุณภาพดี

2.9 ปัญหาการหาค่าเหมาะสมที่มีหลายวัตถุประสงค์

ในการแก้ปัญหาการหาค่าความเหมาะสมที่มีหลายวัตถุประสงค์ (Multi-objective optimization problem) [37] ควรต้องมีการศึกษาทฤษฎีของการหาค่าเหมาะสมที่สุดที่มีหลายวัตถุประสงค์ โดยสามารถสรุปหัวข้อหลักได้ 3 หัวข้อ ดังนี้

2.9.1 หลักการพื้นฐานของการหาค่าเหมาะสมที่สุด

การหาค่าเหมาะสมที่สุด (Optimization) เป็นวิธีการที่ใช้ในการหาคำตอบที่ดีที่สุดของปัญหาภายใต้เงื่อนไขหรือข้อจำกัดที่กำหนดขึ้น การหาค่าที่เหมาะสมที่สุดถือว่าเป็นสิ่งที่ช่วยในการแก้ปัญหาในด้ายวิทยาการคอมพิวเตอร์ ปัญญาประดิษฐ์ การวิจัยการดำเนินงาน และสาขาอื่นที่เกี่ยวข้องได้เป็นอย่างดี ปัญหาการหาค่าที่เหมาะสมที่สุดสามารถแบ่งได้ออกเป็น 2 รูปแบบตามการพิจารณาฟังก์ชันวัตถุประสงค์ (Objective function) หรือฟังก์ชันเป้าหมาย ดังนี้

1. การหาค่าเหมาะสมที่สุดที่พิจารณาฟังก์ชันวัตถุประสงค์เพียงวัตถุประสงค์เดียว (Single objective optimization problem)

2. การหาค่าเหมาะสมที่สุดที่พิจารณาฟังก์ชันวัตถุประสงค์หลายวัตถุประสงค์ (Multi-objective optimization problem) หรือปัญหาการหาค่าความเหมาะสมที่พิจารณาหลายเกณฑ์ (Multi-criteria optimization problem)

2.9.2 ลักษณะของปัญหาการหาค่าเหมาะสมที่สุดที่มีหลายวัตถุประสงค์

การแก้ปัญหาการหาค่าเหมาะสมที่สุดที่มีหลายวัตถุประสงค์เป็นการค้นหาเซตคำตอบภายในพื้นที่ของคำตอบที่เป็นไปได้ เพื่อต้องการหาค่าที่น้อยที่สุด หรือค่ามากที่สุดของฟังก์ชันวัตถุประสงค์ในแต่ละฟังก์ชันพร้อมๆ กัน โดยผลลัพธ์ที่ได้จากการแก้ปัญหา เรียกว่า เซตกลุ่มคำตอบที่ดีที่สุด ปัญหาแบบหลายวัตถุประสงค์ เป็นปัญหาการออกแบบที่มีหลายวัตถุประสงค์ประกอบด้วย m วัตถุประสงค์และตัวแปรตัดสินใจ [38] เขียนอยู่ในรูปทั่วไปดังสมการที่ (2.21)

$$Minimize\ (or\ maximize): \{f_1(x), f_2(x), \dots, f_m(x)\} \tag{2.21}$$

เมื่อ x คือ เวกเตอร์ของตัวแปรตัดสินใจ

$f_i(x)$ คือ ฟังก์ชันวัตถุประสงค์ที่ i เมื่อ $i = 1,2,\dots,m$

ดังนั้นรูปแบบปัญหาการหาค่าเหมาะสมที่สุดที่มีหลายวัตถุประสงค์จะเป็นการค้นหาเวกเตอร์คำตอบ x ภายใต้ m ข้อจำกัดแบบอสมการดังสมการที่ (2.22) หรือภายใต้ n ข้อจำกัดแบบสมการดังสมการที่ (2.23) ซึ่งจะเป็นการกำหนดขอบเขตพื้นที่คำตอบที่เป็นไปได้

$$g_i(x) \leq 0, i = 1,2,\dots,m \tag{2.22}$$

$$h_i(x) = 0, i = 1,2,\dots,n \tag{2.23}$$

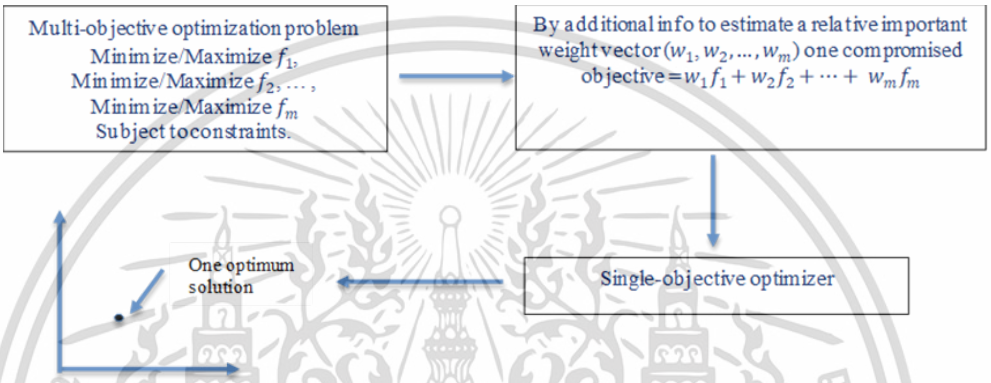
2.9.3 วิธีสำหรับการหาค่าเหมาะสมที่สุดที่มีหลายวัตถุประสงค์

วิธีสำหรับการหาค่าเหมาะสมที่สุดที่มีหลายวัตถุประสงค์ประกอบด้วย 2 วิธีหลักที่ใช้ [39] คือ วิธีผลรวมถ่วงน้ำหนัก (Weighted-sum approach) และวิธีพื้นฐานพาเรโต (Pareto-based approach) โดยกระบวนการของวิธีทั้งสองจะอธิบายดังรูปที่ 2.10 และรูปที่ 2.11 ตามลำดับ

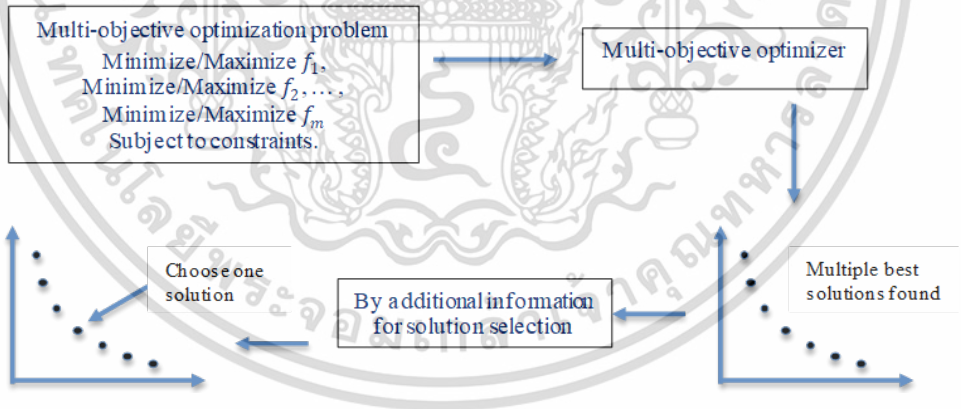
วิธีผลรวมถ่วงน้ำหนักซึ่งเป็นวิธีสำหรับการหาค่าเหมาะสมที่สุดที่มีหลายวัตถุประสงค์แบบดั้งเดิม ทุกวัตถุประสงค์จะจับมารวมกันโดยถ่วงน้ำหนักจนเหลือเพียงวัตถุประสงค์เดียว หลังจากนั้นก็ใช้ตัวหาค่าเหมาะที่สุดวัตถุประสงค์เดียวในการหาคำตอบ ซึ่งโดยส่วนใหญ่ก็จะได้คำตอบเหมาะที่สุดเพียงคำตอบเดียว แต่ก็จะเป็นคำตอบที่ขึ้นอยู่ความคิดส่วนตัวของผู้ใช้เนื่องจากการกำหนดค่าถ่วง

น้ำหนัก แม้ว่าจะมีคำตอบแบบกลางๆ (Compromised solution) ดีที่สุดเพียงคำตอบเดียว เทคนิคนี้ค่อนข้างเป็นที่นิยม เนื่องจากง่ายในการนำไปใช้

สำหรับวิธีพื้นฐานพาเรโตเป็นวิธีหาค่าเหมาะที่สุดวัตถุประสงค์หลายอย่าง คำตอบที่เป็นไปได้เป็นคำตอบที่ไม่มีคำตอบใดดีกว่ากัน หรือไม่มีคำตอบใดที่สามารถครอบงำชุดคำตอบนี้ได้ จะเรียกคำตอบนี้เป็น กลุ่มคำตอบที่เหมาะสมที่สุด (Pareto optimal) และเรียกสมาชิกคำตอบทุกคำตอบที่เหมาะสมที่สุด (Pareto optimal set) หรือเซตคำตอบที่ไม่ถูกครอบงำจากทุกคำตอบ (Non-dominated set)



รูปที่ 2.10 วิธีผลรวมถ่วงน้ำหนัก



รูปที่ 2.11 วิธีพื้นฐานพาเรโต

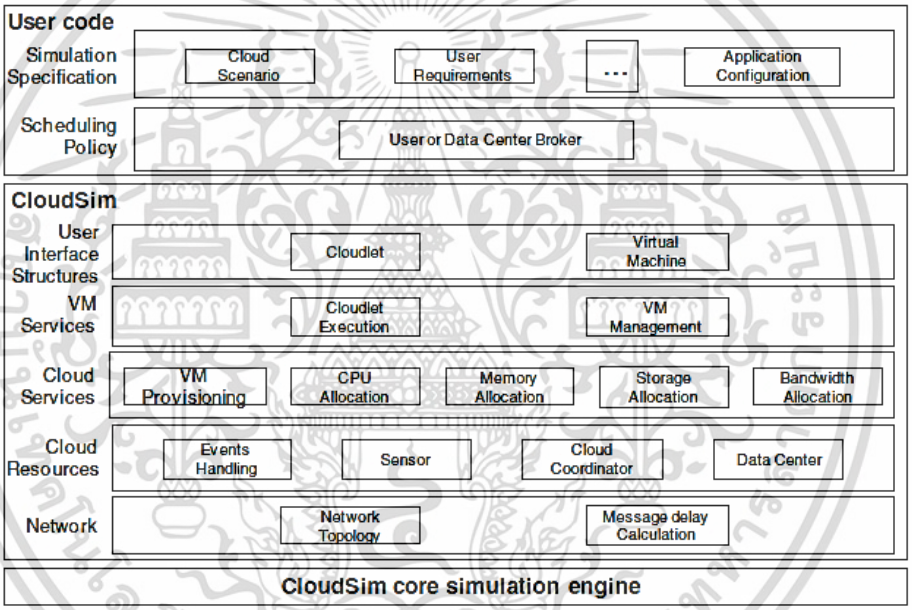
2.10 เครื่องมือที่ใช้ในการสร้างแบบจำลอง CloudSim-3.0.3

CloudSim [40] ถือเป็นเครื่องมือสำหรับนักวิจัยที่ทำการวิจัยเกี่ยวกับการประมวลผลแบบกลุ่มเมฆในแง่มุมเกี่ยวกับโครงสร้างพื้นฐาน (Infrastructure) การจัดการเครือข่าย การใช้พลังงาน (Power) การจัดตารางงานเข้าทำงาน เป็นต้น CloudSim ถูกเขียนขึ้นโดยกลุ่ม Grid Computing and Distributed Systems (GRIDS) Laboratory ซึ่งพัฒนาออกมาในรูปของโปรแกรมการสร้าง

แบบจำลองและเป็นซอฟต์แวร์ระบบเปิด เพื่อให้นักวิจัยสามารถนำไปพัฒนาต่อยอดได้ รายละเอียดของ CloudSim มีดังต่อไปนี้

2.10.1 ลักษณะทั่วไปของ CloudSim

CloudSim [40, 41] เป็นเครื่องมือที่ใช้ในการสร้างแบบจำลองในระบบการประมวลผลแบบกลุ่มเมฆที่มีสภาพแวดล้อมเหมือนจริง อีกทั้งในการสร้างแบบจำลองนั้นยังสามารถทำการปรับเปลี่ยนค่าพารามิเตอร์ตามความต้องการของผู้ใช้ได้ เช่น สามารถจำลองสภาพแวดล้อมของศูนย์ข้อมูล (Data center) ซึ่งประกอบด้วย เครื่องคอมพิวเตอร์เสมือน หน่วยความจำ หน่วยเก็บข้อมูล และแบนด์วิธ (Bandwidth) เป็นต้น จากรูปที่ 2.12 แสดงให้เห็นถึงรูปแบบลำดับชั้นของ CloudSim และสถาปัตยกรรมของ CloudSim



รูปที่ 2.12 สถาปัตยกรรมลำดับชั้นของ CloudSim [40]

2.10.2 คุณสมบัติทั่วไปของ CloudSim

รายละเอียดคุณสมบัติของ CloudSim มีดังต่อไปนี้

1) สนับสนุนการสร้างแบบจำลองที่อยู่บนพื้นฐานของโครงสร้างของระบบการประมวลผลแบบกลุ่มเมฆ เช่นสามารถสร้างศูนย์ข้อมูลบนเครื่องหลัก 1 เครื่องได้ และสามารถจำลองการสร้างเครื่องคอมพิวเตอร์เสมือนได้

2) การทำงานใน CloudSim จะมีแพลตฟอร์มที่เป็นของตนเอง เช่น การจำลองศูนย์ข้อมูล การให้บริการของโบรกเกอร์ (Service brokers) การจัดตารางการทำงาน (Scheduling) และเงื่อนไขในการจัดงานและทรัพยากร (Allocations policies)

3) CloudSim เป็นเครื่องมือที่ถือได้ว่ามีความพร้อมในการจัดการกับเครื่องคอมพิวเตอร์เสมือน เช่น สามารถสร้างและจัดการกับบริการต่างๆ ได้เสมือนเครื่องคอมพิวเตอร์จริง และแต่ละเครื่องคอมพิวเตอร์เสมือนจะทำงานแบบที่เป็นอิสระต่อกัน

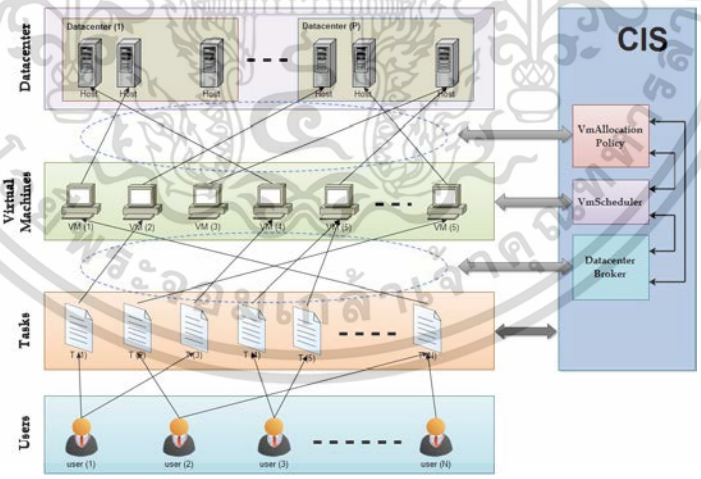
4) CloudSim จะมีความยืดหยุ่นในส่วนของ การจัดสรรการใช้พื้นที่ร่วมกัน (Space share) และการใช้เวลารวมกันในการประมวลผล (Time share) เสมือนการใช้การบริการเครื่องคอมพิวเตอร์เสมือนจริง

2.10.3 รูปแบบการทำงานของ CloudSim

โดยทั่วไปรูปแบบการทำงานในระบบการประมวลผลแบบกลุ่มเมฆจะประกอบด้วยงานจากผู้ใช้จำนวนมากและมีรูปแบบงานที่แตกต่างกัน โดยกำหนดให้

- จำนวนผู้ใช้งานทั้งหมดเท่ากับ m คน ประกอบด้วย $\{User_1, User_2, User_3, \dots, User_m\}$
- จำนวนงานทั้งหมดเท่ากับ n งาน ประกอบด้วย $\{T_1, T_2, T_3, \dots, T_n\}$
- จำนวนเครื่องคอมพิวเตอร์เสมือนเท่ากับ k เครื่อง ประกอบด้วย $\{VM_1, VM_2, VM_3, \dots, VM_k\}$ และ
- จำนวนศูนย์กลางข้อมูล (Datacenter) เท่ากับ p ศูนย์กลางข้อมูล ประกอบด้วย $\{Datacenter_1, Datacenter_2, Datacenter_3, \dots, Datacenter_p\}$

รูปแบบการทำงานของ CloudSim สามารถแสดงได้ดังรูปที่ 2.13



รูปที่ 2.13 การทำงานของ CloudSim [31]

บริการข้อมูลบนคลาวด์ (Cloud information service: CIS) เป็นตัวกลางในการจัดสรรงานระหว่างผู้ใช้บริการและผู้ใช้บริการ โดยในการจัดสรรงานนั้นจะต้องคำนึงถึงความต้องการ

ของผู้ใช้บริการ ซึ่งประกอบด้วย Datacenter broker, VmScheduler และ VmAllocation policy มีรายละเอียดดังนี้

1) Datacenter broker ดูแลในส่วนของการติดต่อระหว่าง ผู้ใช้งาน (User) และผู้ให้บริการ (Service providers) โดยในการจัดสรรงานให้กับ VM นั้นจะต้องคำนึงถึงความต้องการของผู้ใช้งาน

2) VmScheduler ดูแลในส่วนของการจัดสรรทรัพยากรให้กับแต่ละ VM ทั้งนี้ขึ้นอยู่กับการกำหนดการจัดสรรการใช้พื้นที่ร่วมกันและการใช้เวลารวมกันในการประมวลผล

3) VmAllocation policy ทำหน้าที่ในการเลือกศูนย์ข้อมูลให้เหมาะสมกับโฮสต์โดยดูจาก หน่วยความจำ หน่วยเก็บข้อมูลและความต้องการใช้งานของเครื่องคอมพิวเตอร์เสมือน

2.11 งานวิจัยที่เกี่ยวข้อง

การเพิ่มประสิทธิภาพการจัดตารางงานและการทำให้ระบบอยู่ในสภาวะสมดุล (Load balancing) ในสภาพแวดล้อมการประมวลผลแบบกลุ่มเมฆเป็นปัญหาแบบเอ็นพีบริบูรณ์ (NP-hard problem) [42] ดังนั้นจึงทำให้มีงานวิจัยจำนวนมากได้เสนออัลกอริทึมหรือวิธีการเพื่อเพิ่มประสิทธิภาพของประมวลผลแบบกลุ่มเมฆในแง่มุมที่หลายหลาก เช่น การจัดสรรทรัพยากร ขั้นตอนการจัดตารางงานและการทำให้ระบบอยู่ในสภาวะสมดุล เพื่อลดระยะเวลาการทำงานและการลดการใช้พลังงานในระบบ เป็นต้น ทั้งนี้ในการทำงานหากต้องการเข้าใช้งานทรัพยากรที่มีอยู่อย่างมีประสิทธิภาพหรือต้องการความรวดเร็วในการทำงาน จำเป็นต้องมีการจัดตารางงานที่เหมาะโดยคำนึงถึงหลายวัตถุประสงค์หรือสภาพแวดล้อมของคลาวด์ ดังนั้นจึงมีการใช้ขั้นตอนวิธีการฮิวริสติก (Heuristic algorithms) [43, 44] ขั้นตอนวิธีการเมตาฮิวริสติก (Meta-heuristic algorithm) [45] ขั้นตอนวิธีเมตาฮิวริสติกแบบผสม (Hybrid meta-heuristic algorithm) [46-48] หรือแม้กระทั่งวิธีการเรียนรู้ของเครื่อง (Machine learning) [49, 50] เพื่อนำมาการจัดตารางงานและการทำให้ระบบอยู่ในสภาวะสมดุล ตัวอย่างเช่น

ในปี ค.ศ. 2010 Fang *et al.* [51] ได้เสนอการจัดตารางการทำงานในประมวลผลแบบกลุ่มเมฆ โดยคำนึงถึงความต้องการของผู้ใช้งานที่มีการเปลี่ยนแปลงตลอดเวลาและคำนึงถึงการกระจายงานแบบสมดุลในสภาพแวดล้อมของการประมวลผลแบบกลุ่มเมฆ เมื่อผู้ใช้งานและความต้องการในการใช้งานทรัพยากรมีปริมาณเพิ่มมากขึ้น โดยใช้ CloudSim ซึ่งเป็นชุดเครื่องมือที่ในการสร้างแบบจำลองของระบบการประมวลผลแบบกลุ่มเมฆ สามารถใช้ทดสอบประสิทธิภาพของระบบที่พัฒนาขึ้นใหม่ เป็นเครื่องมือที่สะดวกในการกำหนดสภาพแวดล้อมตามที่ผู้ใช้ต้องการ เช่น การตั้งค่าของศูนย์ข้อมูลและเครื่องคอมพิวเตอร์เสมือน เป็นต้น Gan *et al.* [52] มีการเสนอการจัดตารางงานโดยใช้ขั้นตอนวิธีการจำลองการอบเหนียว (Simulated annealing algorithm) โดยมีจุดประสงค์

หลักในการทำงานคือ เพื่อเพิ่มประสิทธิภาพเวลาในการดำเนินการ (Execution time) ของงานในศูนย์ข้อมูลให้เหมาะสม

ในปี ค.ศ. 2015 Patel *et al.* [53] เสนอขั้นตอนวิธีการปรับปรุงโหลดที่สมดุลโดยใช้อัลกอริทึมที่เรียกว่า Enhanced Load balanced Min-Min (ELBMM) algorithm สำหรับจัดตารางงานแบบคงที่ตามวิธีการที่กำหนด (Static scheduling) โดยงานถูกกำหนดให้กับเครื่องคอมพิวเตอร์เสมือน ซึ่งพิจารณาจากเวลาในการดำเนินการและงานจะถูกจัดตารางใหม่เพื่อกระจายไปยังทรัพยากรที่ไม่ได้มีการใช้งาน

การจัดสรรทรัพยากรอย่างมีประสิทธิภาพโดยมุ่งเน้นเพื่อแก้ปัญหาในด้านประสิทธิภาพการใช้พลังงาน (Energy) โดยใช้วิธีการแบบการหาค่าความเหมาะสมที่มีหลายวัตถุประสงค์ ที่เรียกว่า Multi-Objective Optimization (MOO) ถูกกล่าวถึงโดย Shrimali and Patel ในปี ค.ศ. 2017 [54] นอกจากนี้ยังมีการเสนอแนวทางการจัดสรรเครื่องคอมพิวเตอร์เสมือนโดยใช้วิธีการ MOO ผลการทดลองแสดงให้เห็นว่าวิธี MOO นำไปสู่การประหยัดพลังงานเนื่องจากมีการจัดสรรทรัพยากรอย่างมีประสิทธิภาพ โดยไม่กระทบต่อประสิทธิภาพการทำงานของศูนย์ข้อมูล

ต่อมาในปี ค.ศ. 2018 Hussain *et al.* [12] ได้เสนออัลกอริทึมที่เรียกว่า Resource-Aware Load-Balancing Algorithm (RALBA) เพื่อใช้ในการจัดตารางแบบที่สามารถปรับเปลี่ยนได้ตามสภาวะแวดล้อมในการทำงาน (Dynamic scheduling) ที่มีการจัดงานที่มีลักษณะเป็นอิสระต่อกันและไม่สามารถแทรกระหว่างการทำงานให้กับเครื่องคอมพิวเตอร์เสมือน โดยแบ่งการทำงานเป็น 2 ส่วน โดยส่วนแรกคือการเลือกงานที่มีขนาดใหญ่ที่สุดให้กับเครื่องคอมพิวเตอร์เสมือนที่มีประสิทธิภาพสูงที่สุด ส่วนที่ 2 คือ การจัดงานที่เหลือให้กับเครื่องคอมพิวเตอร์เสมือนที่สามารถทำงานได้เร็วที่สุดในการดำเนินการ ผลจากการทดลองพบว่าวิธี RALBA สามารถช่วยลดเวลาในเข้าทำงาน (Makespan) ในระบบ แต่พบว่าวิธีการที่นำเสนอนั้นเกิดปัญหาในเรื่องของการกระจายงานแบบไม่สมดุล (Load imbalance) และมีการใช้งานทรัพยากรที่มีประสิทธิภาพลดลง

ลำดับต่อมาปี ค.ศ. 2019 Zhang *et al.* [55] ได้ศึกษาเกี่ยวกับการประยุกต์ใช้ขั้นตอนวิธีการแบบฮิวริสติกในปัญหาการจัดตารางบนคลาวด์ โดยมีจุดประสงค์ในการจัดตารางงานเพื่อลดเวลาในการทำงานให้เสร็จและลดค่าใช้จ่ายในการดำเนินการ เพื่อเพิ่มประสิทธิภาพในการจัดตารางานในสมาร์ทกริดคลาวด์ (Smart grid cloud)

มีนักวิจัยจำนวนมากได้นำขั้นตอนวิธีการเมตาฮิวริสติกและขั้นตอนวิธีเมตาฮิวริสติกแบบผสมเพื่อนำไปใช้ในแก้ปัญหาในสภาพแวดล้อมของการประมวลผลแบบกลุ่มเมฆ ตัวอย่างงานวิจัย เช่น

ในปี ค.ศ. 2011 Li *et al.*[56] เสนอขั้นตอนวิธีที่เรียกว่า Load balancing Ant Colony Optimization (LBACO) algorithm โดยการนำขั้นตอนวิธีหาค่าเหมาะสมที่สุดด้วยระบบอาณานิคมมด (Ant colony optimization: ACO algorithm) มาใช้ในการจัดตารางการทำงานเพื่อทำให้ระบบเกิดความสมดุลในการทำงานและลดเวลารอเข้าทำงานในระบบ

ในปี ค.ศ. 2013 Tawfeek *et al.* [57] กล่าวถึงการนำขั้นตอนวิธีหาค่าเหมาะสมที่สุดด้วยระบบอาณานิคมมด เพื่อช่วยลดเวลาในการเข้าทำงานของงานในสภาพแวดล้อมของการประมวลผลแบบกลุ่มเมฆ ซึ่งผู้เขียนได้แสดงให้เห็นว่าการเพิ่มประสิทธิภาพของระบบโดยใช้อาณานิคมมดนั้นมีประสิทธิภาพการทำงานดีกว่าวิธีการจัดคิวแบบการวนรอบ (Round Robin: RR algorithm) และวิธีการจัดตารางเวลาแบบมาก่อนได้ก่อน (First come first serve: FCFS algorithm)

ปี ค.ศ. 2015 Awad *et al.* [58] ได้มีการเสนอโมเดลโดยการนำขั้นตอนวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาคมาช่วยในการจัดตารางงานให้กับเครื่องคอมพิวเตอร์เสมือน โดยเสนออัลกอริทึมที่มีชื่อว่า Load Balancing Mutation (balancing) a Particle Swarm Optimization (LBMPSO) โดยอยู่บนพื้นฐานของการจัดตารางและการจัดสรรทรัพยากรในระบบการประมวลผลแบบกลุ่มเมฆ โดยคำนึงถึงเวลาที่ใช้ในการส่งงานไปยังเครื่องคอมพิวเตอร์เสมือน เวลารวมในการทำงาน ค่าใช้จ่ายที่เกิดขึ้นในระบบและความน่าเชื่อถือของการใช้งานระบบคลาวด์

Ali *et al.* [59] ได้เสนอขั้นตอนการจัดตารางการทำงานในสภาพแวดล้อมของการประมวลผลแบบกลุ่มเมฆโดยใช้ขั้นตอนวิธี Dynamic PSO (DAPSO) และขั้นตอนวิธีหาค่าเหมาะสมที่สุดแบบบนกาเหว่า ซึ่งเรียกรวมกันได้ว่า MDAPSO เมื่อทำการเปรียบเทียบประสิทธิภาพการทำงานระหว่าง Dynamic PSO ขั้นตอนวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาคแบบดั้งเดิม และวิธีการที่นำเสนอนั้นโดยการเพิ่มจำนวนงานและเพิ่มจำนวนเครื่องคอมพิวเตอร์เสมือนเข้าไปในระบบ แสดงให้เห็นว่า MDAPSO มีประสิทธิภาพในการทำงานมากกว่าวิธีการอื่นๆ ในแง่ของการลดเวลารวมในการทำงานของระบบและความคุ้มค่าในการใช้ทรัพยากรในการทำงาน

ในปี ค.ศ. 2018 Basu *et al.* (2018) [60] ได้มีการแนะนำให้ใช้ขั้นตอนวิธีเมตาฮิวริสติกแบบผสม ที่เรียกว่า GAACO เกิดจากการผสมระหว่างวิธีเชิงพันธุกรรมและขั้นตอนวิธีหาค่าเหมาะสมที่สุดด้วยระบบอาณานิคมมด เพื่อแก้ปัญหาการจัดตารางงานของแอปพลิเคชันอินเทอร์เน็ตของสรรพสิ่ง (Internet of Things: IoT) ในสภาพแวดล้อมคลาวด์แบบมัลติโปรเซสเซอร์ (Multi-processor) ซึ่งวิธีการที่เสนอนั้นให้การรับรองในเรื่องของการบรรจบกัน (Convergence) ที่เหมาะสม โดยมีการทำการทดสอบกับขนาดของงานและจำนวนโปรเซสเซอร์ที่ต่างกัน ในแง่มุมของเวลารวมในการเข้าทำงานและประสิทธิภาพของระบบ จากการทดลองพบว่า ขั้นตอนวิธี GAACO มีประสิทธิภาพการทำงานดีกว่าวิธีเชิงพันธุกรรมและขั้นตอนวิธีหาค่าเหมาะสมที่สุดด้วยระบบอาณานิคมมด ในสภาพแวดล้อมที่ต่างกันของมัลติโปรเซสเซอร์ที่ต่างกัน

ในปี ค.ศ. 2020 เพื่อลดเวลารวมในการเข้าทำงานในระบบและสร้างสมดุลให้เกิดขึ้นในระบบการประมวลผลแบบกลุ่มเมฆ Kruekaew and Kimpan [61] ได้เสนอขั้นตอนวิธีการจัดตารางงานโดยใช้วิธีการฮิวริสติก (Heuristic task scheduling) รวมกับขั้นตอนวิธีอาณานิคมผึ้งเทียม ที่เรียกว่า HABC algorithm เพื่อจัดตารางการเข้าทำงานของงานและจัดการทรัพยากรระบบคลาวด์ ผลการทดสอบพบว่า การใช้ขั้นตอนอาณานิคมผึ้งเทียมที่นำเสนอทำงานร่วมกับการจัดตารางงานโดยพิจารณาจากขนาดของงานที่มีขนาดใหญ่ที่สุดเข้าทำก่อน (Longest job first) ที่เรียกว่า HABC_LJF

algorithm ให้ประสิทธิภาพในการจัดตารางการทำงานเหมาะสมที่สุด ในการกระจายงานแบบสมดุล และลดเวลารวมในการทำงาน เมื่อปริมาณงานในระบบเพิ่มมากขึ้นและสภาพแวดล้อมระบบมีการเปลี่ยนแปลง

ตัวอย่างของงานวิจัยเกี่ยวกับปัญหาการจัดตารางงานที่มีหลายวัตถุประสงค์ในการประมวลผลแบบกลุ่มเมฆถูกแก้ไขโดยใช้ขั้นตอนวิธีการเมตาฮิวริสติกและขั้นตอนวิธีเมตาฮิวริสติกแบบผสม เช่น

ในปี ค.ศ. 2015 Zuo *et al.* [46] ได้เสนอวิธีการจัดตารางงานที่มีหลายวัตถุประสงค์ โดยคำนึงถึงเรื่องของประสิทธิภาพของระบบและค่าใช้จ่ายที่เกิดขึ้น โดยใช้ขั้นตอนวิธีระบบอาณานิคมมดเพื่อหาวิธีการที่เหมาะสมที่สุด ในการทดลองได้ทำการเปรียบเทียบกับขั้นตอนวิธี Min-Min และวิธีการจัดตารางเวลาแบบมาก่อนได้ก่อน จากการทดสอบพบว่าวิธีการที่นำเสนอนั้นมีประสิทธิภาพที่ดีกว่าวิธีการอื่นที่นำมาเปรียบเทียบ

ต่อมาในปี ค.ศ. 2016 He *et al.* [62] ได้เสนออัลกอริทึมที่เรียกว่า Adaptive Multi-objective Task Scheduling (AMTS) โดยคำนึงถึงการใช้ทรัพยากร (Resource utilization) ต้นทุน (Cost) การใช้พลังงาน (Energy consumption) และเวลาในการเข้าทำงานของงาน ซึ่งได้มีการนำขั้นตอนวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาคเพื่อช่วยในการจัดตารางงานที่มีหลายวัตถประสงค์ โดยวิธีการที่นำเสนอนั้นได้มีการใช้ค่าสัมประสิทธิ์เพื่อช่วยในการปรับความเร็วและเพิ่มความหลากหลายในการทำงานของอนุภาค จากการทดสอบประสิทธิภาพของวิธีการที่นำเสนอพบว่าสามารถหาวิธีการแก้ปัญหาการจัดตารางงานบนคลาวด์ได้เหมาะสมที่สุด

ในปี ค.ศ. 2017 ได้มีการเสนออัลกอริทึมการหาค่าที่เหมาะสมของวาฬ (Whale optimization algorithm: WOA algorithm) แบบหลายวัตถุประสงค์ในสภาพแวดล้อมของคลาวด์ โดย Reddy and Kumar [63] ซึ่งทางผู้เขียนพยายามจัดตารางงานโดยอาศัยค่าความเหมาะสม (Fitness) โดยอิงตามเงื่อนไข 3 ประการคือ คุณภาพของการบริการ (Quality of service) การใช้พลังงาน และการใช้ทรัพยากร ซึ่งเมื่อพิจารณาจากพารามิเตอร์ที่กำหนดพบว่า สามารถลดเวลาทำงานและค่าใช้จ่ายในการเข้าใช้เครื่องคอมพิวเตอร์เสมือนได้

ในปี ค.ศ. 2021 Alsadie [64] เสนอเฟรมเวิร์กของขั้นตอนวิธีการเมตาฮิวริสติกสำหรับการจัดสรรเครื่องคอมพิวเตอร์เสมือนแบบไดนามิกด้วยการจัดตารางงานที่เหมาะสมในสภาพแวดล้อมการประมวลผลแบบกลุ่มเมฆ ที่เรียกว่า MDVMA โดยพิจารณาการทำงานโดยคำนึงถึงการจัดตารางงานที่มีหลายวัตถุประสงค์ ซึ่งมีการนำอัลกอริทึมเชิงพันธุกรรมแบบการจัดลำดับที่ไม่ถูกครอบงำ (Non-dominated Sorting Genetic Algorithm II: NSGA-II) เข้ามาช่วยในการทำงาน เพื่อช่วยลดการใช้พลังงาน (Energy usage) เวลารวมในการทำงานและค่าใช้จ่ายในการทำงาน ในการทดลองได้ทำการทดสอบกับชุดข้อมูลที่เรียกว่า Heterogeneous Computing Scheduling Problems (HCSP) และมีการจำลองสภาพแวดล้อมของระบบโดยใช้ CloudSim จากการทำสอบประสิทธิภาพพบว่า MDVMA algorithm สามารถช่วยเพิ่มประสิทธิภาพในการจัดตารางงานได้ดีกว่าการใช้ขั้นตอนวิธี

อาณานิคมผึ้งเทียม อัลกอริทึมการหาค่าที่เหมาะสมของวาฬ และขั้นตอนวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค ในแง่ของการลดการใช้พลังงาน เวลาในการทำงาน และค่าใช้จ่ายในศูนย์ข้อมูล

ในปีเดียวกัน ค.ศ. 2021 Guo [47] ได้เสนอการเพิ่มประสิทธิภาพในการจัดตารางเวลางานที่มีหลายวัตถุประสงค์ในการประมวลผลแบบกลุ่มเมฆ โดยใช้วิธี Fuzzy self-defense โดยวิธีการที่นำเสนอมีประสิทธิภาพในแง่ของเวลาเสร็จสิ้นสูงสุด (Maximum completion time) อัตราการละเมิดกำหนดเวลา (Deadline violation rate) และการใช้ทรัพยากรเครื่องคอมพิวเตอร์เสมือน จากการทดสอบประสิทธิภาพพบว่า วิธีการที่นำเสนอมีประสิทธิภาพมากกว่าวิธีการจัดตารางที่มีหลายวัตถุประสงค์ในการประมวลผลแบบกลุ่มเมฆโดยใช้ขั้นตอนวิธีระบบอาณานิคมมด [35] และการจัดตารางการทำงานโดยใช้วิธีการเรียนรู้แบบเสริมกำลัง

ในปี ค.ศ. 2020 Sanaj and Prathap [48] เสนอการจัดตารางงาน โดยใช้วิธีเชิงพันธุกรรมและอัลกอริทึมการหาค่าที่เหมาะสมของวาฬ ที่เรียกว่า MRQFLDA algorithm ในขั้นแรกเสนอให้ทำการแยกคุณสมบัติที่เกี่ยวข้องกันของงาน หลังจากนั้นจึงใช้ MRQFLDA algorithm ในการลดคุณสมบัติที่สกัดที่แยกออกมา และทำการจัดกลุ่มงานย่อย โดยใช้ MapReduce framework และใช้วิธีเชิงพันธุกรรม และอัลกอริทึมการหาค่าที่เหมาะสมของวาฬในการกำหนดงานย่อยให้กับเครื่องคอมพิวเตอร์เสมือน จากการทดลองพบว่าวิธีการที่นำเสนอสามารถทำงานได้ดีกว่าวิธีการอื่น

ในปี ค.ศ. 2018 Amini et al. [65] ได้เสนอกระบวนการจัดสรรทรัพยากรสำหรับเครื่องคอมพิวเตอร์เสมือนในระบบการประมวลผลแบบกลุ่มเมฆ โดยใช้ขั้นตอนวิธีการเพิ่มประสิทธิภาพของแมลงปอ (Dragonfly optimization algorithm) จากการทดลองแสดงให้เห็นว่าการใช้ขั้นตอนวิธีการเพิ่มประสิทธิภาพของแมลงปอสามารถช่วยปรับปรุงการวิธีการจัดตารางงาน การช่วยให้ระบบสามารถกระจายงานแบบสมดุล และช่วยในการจัดสรรทรัพยากรได้ดีกว่าการใช้ขั้นตอนอาณานิคมมดและขั้นตอนการผสมระหว่างขั้นตอนวิธีการหาคำตอบเหมาะสมที่สุดแบบกลุ่มอนุภาคและขั้นตอนอาณานิคมมด (ACO-PSO)

ต่อมาในปี ค.ศ. 2019 Madni et al. [66] เสนอขั้นตอนวิธีการหาค่าเหมาะสมที่สุดแบบนกกาเหว่าแบบหลายวัตถุประสงค์ (Multi-objective Cuckoo Search Optimization: MOCSO algorithm) สำหรับปัญหาการจัดตารางงานในสภาพแวดล้อมการประมวลผลแบบกลุ่มเมฆ โดยมีเป้าหมายหลักเพื่อลดต้นทุนสำหรับผู้ใช้บริการการประมวลผลแบบกลุ่มเมฆและช่วยลดเวลาในการทำงานในระบบ ผลจากการทดลองแสดงให้เห็นว่า วิธี MOCSO มีประสิทธิภาพการทำงานที่เหนือกว่าวิธีการหาคำตอบเหมาะสมที่สุดโดยใช้ขั้นตอนอาณานิคมมดแบบหลายวัตถุประสงค์ (Multi-objective ACO: MOACO) วิธีการหาคำตอบที่เหมาะสมโดยใช้ขั้นตอนวิธีเชิงพันธุกรรมแบบหลายวัตถุประสงค์ (Multi-objective GA: MOGA) การหาคำตอบโดยใช้ค่าต่ำสุดสูงสุด (Multi-objective Min-Max: MOMM) และขั้นตอนวิธีการหาคำตอบเหมาะสมที่สุดแบบกลุ่มอนุภาคแบบหลายวัตถุประสงค์ (Multi-objective PSO: MOPSO)

ในปี ค.ศ. 2019 Pang et al. [67] พัฒนาวิธีการจัดตารางงานโดยรวมการประมาณค่าอัลกอริทึมการกระจาย (Estimation of distribution algorithms) และวิธีการหาคำตอบที่เหมาะสมโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม (GA) โดยเรียกวิธีการนี้ว่า EDA-GA วิธีการที่เสนอถูกเปรียบเทียบกับวิธีการประมาณค่าอัลกอริทึมการกระจายและวิธีการหาคำตอบที่เหมาะสมโดยใช้ขั้นตอนวิธีเชิงพันธุกรรม โดยใช้ CloudSim ในการจำลองสภาพแวดล้อมการประมวลผลแบบกลุ่มเมฆ ผลจากการทดลองแสดงให้เห็นว่า EDA-GA สามารถช่วยลดเวลาทำงานและช่วยปรับปรุงความสามารถในการกระจายงานไปยังทรัพยากรได้อย่างทั่วถึง

ต่อมาในปี ค.ศ. 2020 Neelima and Reddy [68] ได้เสนอขั้นตอนวิธีการจัดตารางงานในการประมวลผลแบบกลุ่มเมฆ โดยใช้ขั้นตอนวิธีการเพิ่มประสิทธิภาพของแมลงปอและอัลกอริทึมหิ่งห้อย (Firefly algorithm) โดยคำนึงถึงฟังก์ชันหลายวัตถุประสงค์ได้แก่ เวลาในการทำงาน ค่าใช้จ่ายในการดำเนินงานและภาระงานของระบบ ผลจากทดสอบประสิทธิภาพพบว่าวิธีการที่เสนอสามารถช่วยกระจายงานไปยังทรัพยากรได้ดี

ในปี ค.ศ. 2019 Gamal et al. [69] นำเสนออัลกอริทึมการออสโมซิสระหว่างขั้นตอนอาณานิคมผึ้งเทียมและขั้นตอนอาณานิคมมด โดยเรียกวิธีการนี้ว่า OH_BAC ซึ่งเกิดขึ้นจากการใช้ทฤษฎีออสโมซิสในทางวิทยาศาสตร์เคมี โดยอัลกอริทึมนี้สำหรับค้นหาการปรับสมดุลโหลดสำหรับการจัดวางเครื่องคอมพิวเตอร์เสมือน ผลจากการทดลองพบว่า OH_BAC สามารถช่วยปรับปรุงการใช้พลังงาน จำนวนการย้ายการทำงานของเครื่องคอมพิวเตอร์เสมือนและจำนวนการที่โฮสต์หยุดทำงานได้ดี

การใช้วิธีการทางการเรียนรู้ของเครื่องถูกนำมาแก้ปัญหาที่ท้าทายในสภาพแวดล้อมการประมวลผลแบบกลุ่มเมฆ [70] ตัวอย่างของงานวิจัยเช่น

ในปี ค.ศ. 2014 Farahnakian *et al.* [49] ได้เสนอวิธีการเรียนรู้แบบเสริมกำลังเพื่อลดการใช้พลังงานและเพิ่มประสิทธิภาพการใช้ทรัพยากรในศูนย์ข้อมูลบนคลาวด์

ในปี ค.ศ. 2019 Rugwiro and Ding. [50] ได้เสนอการจัดตารางงานและแบบจำลองการจัดสรรทรัพยากรโดยการผสมระหว่างขั้นตอนวิธีระบบอาณานิคมมดและวิธีการเรียนรู้แบบเสริมกำลังเชิงลึก (Deep reinforcement learning: DRL) โดยมีเป้าหมายเพื่อลดเวลาการทำงานรวมและปรับปรุงการใช้ทรัพยากรที่ไม่ได้ใช้งาน (Idle) โดยใช้การท่องเข้าไปในไบนารีทรีแบบ Inorder (Binary inorder traversal tree) โดยค่าถ่วงน้ำหนัก (Weighted values) สำหรับการจัดตารางงานโดยใช้วิธีการเรียนรู้แบบเสริมกำลังเชิงลึกเพื่อค้นหาทรัพยากรที่ไม่มีการใช้งาน และใช้ขั้นตอนวิธีระบบอาณานิคมมดสำหรับค้นหาเครื่องคอมพิวเตอร์เสมือนที่เหมาะสมที่สุดสำหรับงานนั้นๆ

ในปี ค.ศ. 2020 Jena *et al.* [71] เสนอขั้นตอนวิธีการเมตาฮิวริสติกที่เรียกว่า QMPSO algorithm สำหรับปรับสมดุลของโหลดระหว่างเครื่องคอมพิวเตอร์เสมือนในการประมวลผลแบบกลุ่มเมฆ โดยใช้การผสมระหว่างการปรับปรุงขั้นตอนวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค (Modified particle swarm optimization: MPSO) และพัฒนาการใช้ขั้นตอนวิธีการเรียนรู้แบบคิว (Q-

learning algorithm) จากการทดสอบประสิทธิภาพของ QMPSO algorithm พบว่าช่วยปรับปรุงในเรื่องของเวลารวมในการเข้าทำงาน เพิ่มปริมาณงานที่ทำได้ในช่วงเวลาหนึ่ง (Throughput) และลดเวลารอการเข้าทำงานในระบบ และในปีเดียวกัน Caviglione *et al.* [72] แนะนำการจัดวางเครื่องคอมพิวเตอร์เสมือนในศูนย์ข้อมูลบนคลาวด์โดยใช้วิธีการเรียนรู้แบบเสริมกำลังเชิงลึก เพื่อเลือกตำแหน่งที่เหมาะสมที่สุดที่เป็นไปได้สำหรับแต่ละเครื่องคอมพิวเตอร์เสมือน

บทที่ 3
วิธีการดำเนินงานวิจัย

ในงานวิจัยนี้เสนอการประยุกต์ใช้ปัญญาประดิษฐ์สำหรับปัญหาการจัดตารางงานที่มีหลายวัตถุประสงค์ (Multi-objective optimization problem) ในการประมวลผลแบบกลุ่มเมฆ โดยใช้ขั้นตอนอาณานิคมผึ้งเทียมกับวิธีการเรียนรู้แบบเสริมแรง โดยใช้ขั้นตอนวิธีการเรียนรู้แบบคิว (Q-learning) เรียกอัลกอริทึมที่เสนอนี้ว่า "MOABCQ" อีกทั้งมีการพิจารณาลำดับการเข้าทำงานให้เหมาะสมกับสภาพแวดล้อมของทรัพยากรที่มีอยู่ เพื่อหารูปแบบการจัดตารางงานที่เหมาะสมที่สุดอีกด้วย

3.1 นิยามปัญหาและการกำหนดฟังก์ชันวัตถุประสงค์

ในการเปรียบเทียบคำตอบของปัญหาการหาค่าความเหมาะสมที่สุดที่มีหลายวัตถุประสงค์ใช้วิธีการหาค่าความเหมาะสมแบบผลรวมถ่วงน้ำหนัก เพื่อหาคำตอบที่เหมาะสมที่สุด มีรายละเอียดดังต่อไปนี้

ปัญหาการจัดตารางงานสามารถนำเสนอได้ดังนี้

Input:

- กำหนดให้ $V = \{v_1, v_2, v_3, \dots, v_m\}$ โดยที่ V คือชุดของเครื่องคอมพิวเตอร์เสมือน (VMs) และ m คือ จำนวนเครื่องคอมพิวเตอร์เสมือนทั้งหมดในเครือข่ายคลาวด์ และแต่ละเครื่องมีทรัพยากรเป็นของตัวเอง (เช่น ความเร็วของหน่วยประมวลผลกลาง (CPU speed: CS) ความเร็วของหน่วยความจำหลัก (RAM speed: RS) และแบนด์วิธ (BW)) โดยที่ v_i หมายถึง เครื่องลำดับที่ i^{th} VM
- กำหนดให้ $T = \{t_1, t_2, t, \dots, t_n\}$ โดยที่ T คือ ชุดของงานที่กำหนด และ n คือ จำนวนงานที่มีอิสระทั้งหมดในระบบที่ดำเนินการในเครื่องคอมพิวเตอร์เสมือนจำนวน m โดยที่ t_j หมายถึงลำดับงานที่ j^{th} แต่ละงานที่ส่งมาประกอบด้วยจำนวนคำสั่ง (Number of instructions: TS) จำนวนหน่วยความจำที่ร้องขอ (Memory required: MR) และจำนวนหน่วยประมวลผลการที่มีการร้องขอ (CPU required: CR) เป็นต้น โดยรูปแบบการเข้าทำงานในระบบเป็นการทำงานแบบไม่สามารถขัดจังหวะหรือแทรกกลางคันได้ (Non-Preemptive Process)

Output:

- มีการจัดตารางที่เหมาะสมให้กับงานจำนวน n งาน และเครื่องคอมพิวเตอร์เสมือนจำนวน m เครื่อง เพื่อให้สามารถลดเวลารวมในการเข้าทำงาน (Makespan) ลดค่าใช้จ่ายที่เกิดขึ้น

จากการเข้าใช้งานเครื่องคอมพิวเตอร์เสมือน (Cost) และเพิ่มประสิทธิภาพการเข้าใช้งานทรัพยากรให้มากที่สุด (Resource utilization) อีกทั้งยังช่วยให้มีการกระจายการใช้ทรัพยากรแบบสมดุล

ในงานวิจัยนี้เสนออัลกอริทึมสำหรับการจัดตารางงาน โดยใช้เงื่อนไขในการจัดตารางที่มีหลายวัตถุประสงค์ เพื่อเพิ่มประสิทธิภาพในการทำงานของระบบ โดยเงื่อนไขที่นิยมใช้ในการพิจารณาความเหมาะสมในการจัดตารางงาน (Evaluation factors) ได้แก่ ค่าใช้จ่าย (Cost) พลังงาน (Energy consumption) เวลารวมในการทำงาน (Task completion time) เวลารอในการเข้าทำงาน (Task waiting time) อัตราการเสียหาย (Failure rate) และเวลาในการเข้าทำงาน และความน่าเชื่อถือ (Reliability) เป็นต้น [3] โดยอัลกอริทึมที่นำเสนอจะพิจารณาเงื่อนไขความเหมาะโดยใช้ค่าใช้จ่ายที่เกิดขึ้นจากการเข้าใช้งาน การใช้งานทรัพยากรให้มีประสิทธิภาพ และเวลารวมในการเข้าทำงาน เป็นปัจจัยที่ใช้ในการช่วยในการจัดตารางงานที่เหมาะสม การจัดตารางและสร้างโหลดสมดุลในสภาพแวดล้อมการประมวลผลแบบกลุ่มเมฆจำเป็นต้องมีการออกแบบฟังก์ชันวัตถุประสงค์ (Fitness function) เพื่อให้ได้คำตอบให้เหมาะสมที่สุด อีกทั้งยังต้องคำนึงถึงเงื่อนไขที่ใช้ในการหาคำตอบ เงื่อนไขที่ใช้ในการพิจารณาสามารถแบ่งออกได้ 3 วัตถุประสงค์ (Objectives) ดังนี้ คือ

วัตถุประสงค์ที่ 1: กำหนดในเงื่อนไขของเวลารวมในการเข้าทำงาน (Makespan) ซึ่งหมายถึงเวลาที่ระบบทำงานสุดท้ายเสร็จสิ้น เวลารวมในการเข้าทำงานถือเป็นปัจจัยที่ได้รับความนิยมสำหรับการจัดตารางงานแบบหลายวัตถุประสงค์ ซึ่งจะช่วยลดเวลาในการดำเนินการและยังช่วยให้สามารถทำงานได้เสร็จสิ้นก่อนเวลาที่กำหนด ในการทำงานของเครื่องคอมพิวเตอร์เสมือนแต่ละเครื่องจะมีเวลาดำเนินการเพื่อให้ทำงานให้เสร็จสิ้นที่แตกต่างกัน ซึ่งเวลาที่ระบบทำงานสุดท้ายเสร็จสิ้นจะเป็นตัวกำหนดเวลาทำงานของเครื่องคอมพิวเตอร์เสมือนในระบบ โดยพบว่าหากค่าเวลารวมในการทำงานที่มากที่สุด (Maximum the execution time) มีค่าสูงมากส่งผลให้ค่าเวลารวมในการเข้าทำงานมีค่ามากตามไปด้วย ถือว่าระบบนั้นมีการกระจายงานไปยังเครื่องคอมพิวเตอร์เสมือนได้ไม่ดี ในทางตรงกันข้าม หากค่าเวลารวมในการทำงานที่มากที่สุดมีค่าน้อยย่อมส่งผลให้ค่าเวลารวมในการเข้าทำงานมีค่าน้อยตามไปด้วย ซึ่งหมายความว่าระบบมีการกระจายงานไปยังทรัพยากรในระบบอย่างทั่วถึง

โดยที่แต่ละงาน (task) แทนด้วย t_j ($t_j \in$ T) ถูกกำหนดให้เครื่องคอมพิวเตอร์เสมือนแทนด้วย v_i ($v_i \in$ V) ถูกแทนด้วย t_{ji} ดังนั้น งานภายในเครื่องคอมพิวเตอร์เสมือนแทนด้วย $v_i = \{t_{xi}, t_{yi}, \dots, t_{zi}\}$

สามารถหาเวลารวม (Execution time: ET) ของงานที่ทำงานในเครื่องคอมพิวเตอร์เสมือนลำดับที่ v_i ได้จากสมการที่ (3.1)

$$ET(v_i) = \sum_{t_{ji} \in v_i} ExtTime(t_{ji}) = \frac{\sum_{t_{ji} \in v_i} length(t_j)}{CPU(v_i)} \quad (3.1)$$

โดยที่ $ExtTime(t_{ji})$ คือ เวลารวมในการทำงานของ t_j ที่มีการทำงานในเครื่องคอมพิวเตอร์เสมือนที่ v_i สามารถคำนวณได้จากสมการที่ (3.2)

$$ExtTime(t_{ji}) = \frac{length(t_j)}{CPU(v_i)} \quad (3.2)$$

โดยที่ $length(t_j)$ คือขนาดของงานลำดับที่ j และ $CPU(v_i)$ คือ อัตราการประมวลผลของหน่วยประมวลผลกลาง (CPU rate) ที่ใช้ในการประมวลผลของเครื่องคอมพิวเตอร์เสมือนลำดับที่ j^{th} ในคลาวด์ Makespan คือ ค่าเวลาในการดำเนินงานที่มีค่ามากที่สุดของเครื่องคอมพิวเตอร์เสมือนทั้งหมด สามารถคำนวณได้จากสมการที่ (3.3)

$$Makespan = Max(ExtTime(v_i)), 1 \leq i \leq m \quad (3.3)$$

กำหนดให้ $MinMakespan$ คือ ขอบล่าง (Lower bound) ของค่าเวลาในการดำเนินงานที่เวลาสั้นที่สุดที่ระบบต้องการเพื่อทำงานทั้งหมดให้เสร็จสิ้น $MinMakespan$ สามารถคำนวณได้จากสมการที่ (3.4)

$$MinMakespan = Min(ExtTime(v_i)), 1 \leq i \leq m \quad (3.4)$$

ฟังก์ชันวัตถุประสงค์ในแง่ของค่าเวลาในการดำเนินงานที่มีค่ามากที่สุด (Makespan) (F_1) สามารถคำนวณได้จากสมการที่ (3.5)

$$F_1 = \frac{MinMakespan}{Makespan} \quad (3.5)$$

วัตถุประสงค์ที่ 2: ถูกกำหนดในแง่ของค่าใช้จ่าย (Cost) คือ ค่าใช้จ่ายที่ใช้ในการร้องขอเข้าทำงานของงานนั้น โดยสามารถคำนวณต้นทุนทั้งหมดที่เกิดขึ้นได้จาก เวลาและค่าใช้จ่ายที่เกิดขึ้นจากการใช้หน่วยประมวลผลกลาง เวลาและค่าใช้จ่ายที่เกิดขึ้นจากการใช้หน่วยความจำ เวลาและค่าใช้จ่ายที่เกิดขึ้นจากการใช้งานแบนด์วิธ สามารถคำนวณค่าใช้จ่ายที่เกิดขึ้นโดยประมาณเมื่อ t_j เข้าทำงานที่ v_i ได้จากสมการที่ (3.6)

$$Cost(t_{ji}) = (c_1 * ExtTime(t_{ji})) + (c_2 * ExtTime(t_{ji})) + (c_3 * ExtTime(t_{ji})) \quad (3.6)$$

โดยที่ c_1, c_2, c_3 คือ ค่าใช้จ่ายในการใช้หน่วยประมวลผลกลางต่อหน่วย (CPU usage cost per unit) ค่าใช้จ่ายในการใช้หน่วยความจำต่อหน่วย (Memory usage cost per unit) และค่าใช้จ่ายในการใช้แบนด์วิธต่อหน่วย (Bandwidth usage cost per unit) ในเครื่องคอมพิวเตอร์เสมือนที่ v_i ตามลำดับ

ผลรวมทั้งหมดของค่าใช้จ่าย $(TCost)$ คำนวณได้จากผลรวมของงานทั้งหมดที่เข้าทำงานในเครื่องคอมพิวเตอร์เสมือนทั้งหมด โดยสามารถคำนวณได้จากสมการที่ (3.7)

$$TCost = \sum_{j=1}^{n} \sum_{i=1}^{m} Cost(t_{ji}) \quad (3.7)$$

กำหนดให้ $MinTCost$ คือ ค่าใช้จ่ายที่มีค่าน้อยที่สุด เมื่อชุดงาน T เข้าทำงานในเครื่องคอมพิวเตอร์เสมือน โดยที่เครื่องคอมพิวเตอร์เสมือนนั้นดำเนินงาน t_j แล้วให้ค่าใช้จ่ายที่เกิดขึ้นมีค่าน้อยที่สุด จะเรียกว่าค่านั้นว่า $MinTCost(t_j)$ ซึ่งหมายความว่า $MinTCost$ เฉพาะงานที่ t_j เท่านั้น สามารถคำนวณได้จากสมการที่ (3.8)

$$MinTCost = \sum_{t_j \in T} MinCost(t_j) = \sum_{t_{ji} \in T} min_{1 \le i \le m}(Cost(t_{ji})) \quad (3.8)$$

ฟังก์ชันวัตถุประสงค์ในแง่ของต้นทุนหรือค่าใช้จ่ายที่เกิดขึ้น (Cost) ทั้งหมด (F_2) สามารถคำนวณได้จากสมการที่ (3.9)

$$F_2 = \frac{MinTCost}{TCost} \quad (3.9)$$

วัตถุประสงค์ที่ 3: ถูกกำหนดในแง่ของการใช้งานทรัพยากร (Utilization of resources) ที่ส่งไปยังหน่วยประมวลผลที่แตกต่างกับในคลาวด์ โดยในที่นี้คำนึงถึงหน่วยประมวลผลกลางและหน่วยความจำ ซึ่งหากมีการส่งงานที่มีการร้องขอเข้าทำงานในเครื่องคอมพิวเตอร์เสมือนที่ v_i สามารถคำนวณโหลดของหน่วยความจำของเครื่องคอมพิวเตอร์เสมือนที่ v_i (LM_i) ได้จากสมการที่ (3.10)

$$LM_i = AM_i + \frac{RM_j}{TM_i} \quad (3.10)$$

โดยที่ AM_i คือ จำนวนการใช้หน่วยความจำก่อนดำเนินการงาน t_j ที่ i^{th} VM

RM_j คือ จำนวนหน่วยความจำที่มีการร้องขอของงานลำดับที่ j^{th}

TM_i คือ หน่วยความจำทั้งหมดที่มีอยู่ (Total memory available) ที่ i^{th} VM

พารามิเตอร์ตัวต่อมาคือ หน่วยประมวลผลกลาง ซึ่งหากมีการส่งงานที่มีการร้องขอเข้าทำงานในเครื่องคอมพิวเตอร์เสมือนที่ v_i สามารถคำนวณโหลดของหน่วยประมวลผลกลางของเครื่องคอมพิวเตอร์เสมือนที่ $v_i(LC_i)$ ได้จากสมการที่ (3.11)

$$LC_i = AC_i + \frac{RC_j}{TC_i} \quad (3.11)$$

โดยที่ AC_i คือ จำนวนการใช้หน่วยประมวลผลกลางก่อนดำเนินการงาน t_j ที่ i^{th} VM

RC_j คือ จำนวนของหน่วยประมวลผลกลางที่มีการร้องขอของงานลำดับที่ j^{th}

TC_i คือ จำนวนของหน่วยประมวลผลกลางทั้งหมดที่มีอยู่ (Total CPU available) ที่ i^{th} VM

การประเมินการใช้เครื่องคอมพิวเตอร์เสมือน (VU_i) [73] สามารถคำนวณได้จากสมการที่ (3.12)

$$F_3 = VU_i = \frac{\omega_1}{1-LM_i} * \frac{\omega_2}{1-LC_i} \tag{3.12}$$

โดยที่ ω_1, ω_2 คือค่าถวงน้ำหนัก (Weight) ของหน่วยประมวผลกลางและหน่วยความจำตามลำดับ โดยที่ $\omega_1 + \omega_2 = 1$ ในงานวิจัยนี้กำหนดให้ ω_1, ω_2 มีค่าเท่ากับ 0.5 ทั้งคู่ เนื่องจากให้ความสำคับของหน่วยประมวผลกลางและหน่วยความจำเท่ากัน

โหลดทั้งหมดบน k โฮสต์ (LH) เมื่อ k คือจำนวนโฮสต์ทั้งหมดในระบบ สามารถคำนวณได้จากสมการที่ (3.13)

$$LH_k = \sum_{i=0}^{m_k} VU_{ki} \tag{3.13}$$

โหลดเฉลี่ยของเครื่องคอมพิวเตอร์ (Physical machines) ทั้งหมดในคลาวด์ (AL) สามารถคำนวณได้จากสมการที่ (3.14) โดยที่ p คือ จำนวนของโฮสต์ในเครือข่ายคลาวด์

$$AL = \frac{\sum_{k=0}^{p} LH_k}{p} \tag{3.14}$$

ความแตกต่างของโหลดระหว่างแต่ละโฮสต์และโหลดเฉลี่ยบนคลาวด์ (Average load on Cloud network) คำนวณได้จาก $|LH_k - AL|$ ฟังก์ชันวัตถุประสงค์ถูกกำหนดในแง่ของการใช้งานทรัพยากรสามารถคำนวณได้จากสมการที่ (3.15)

$$F_3 = \sum_{k=0}^{p} |LH_k - AL| \tag{3.15}$$

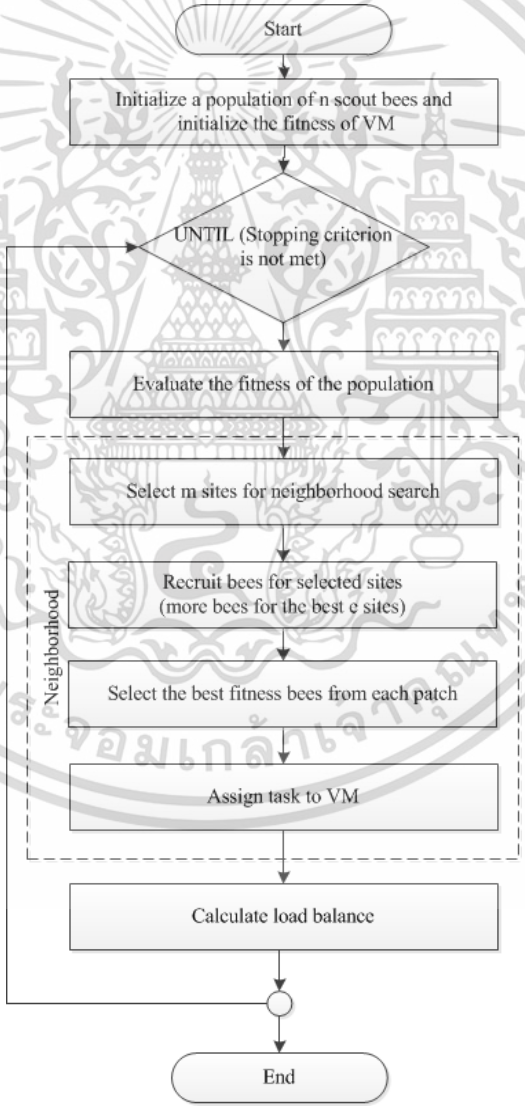
ฟังก์ชันวัตถุประสงค์ (F) ถูกสร้างขึ้นโดยการหาค่าผลรวมถ่วงน้ำหนัก เพื่อหาคำตอบที่เหมาะสมที่สุด โดย F ที่นำเสนอแสดงดังสมการที่ (3.16)

$$F = (\gamma_1 * F_1) + (\gamma_2 * F_2) + (\gamma_3 * F_3) \tag{3.16}$$

โดยที่ γ_1, γ_2 และ γ_3 $(\gamma \epsilon [0,1])$ คือค่าสัมประสิทธิ์ความสมดุลระหว่างค่าเวลาในการดำเนินงานที่มีค่ามากที่สุด ค่าใช้จ่ายที่ใช้ในการดำเนินงานและการใช้งานทรัพยากร โดยที่ F ที่มีค่าสูงสุดถือได้ว่าเป็นผลลัพธ์ที่มีความเหมาะสมมากที่สุด

3.2 การประยุกต์ใช้ขั้นตอนอาณานิคมผึ้งเทียม (Artificial Bee Colony Algorithm)

ขั้นตอนวิธีการจัดตารางงานในการเข้าใช้เครื่องคอมพิวเตอร์เสมือน เพื่อให้มีการกระจายงานในระบบสมดุล โดยประยุกต์ใช้ขั้นตอนวิธีอาณานิคมผึ้งเทียมควบคู่กับวิธีการทางฮิวริสติก โดยเรียกอัลกอริทึมที่ได้นี้ว่า “HABC” สามารถแสดงได้ดังรูปที่ 3.1



รูปที่ 3.1 ผังงานแสดงการจัดตารางงานเพื่อเข้าใช้เครื่องคอมพิวเตอร์เสมือนและระบบมีการกระจายงานแบบสมดุล โดยประยุกต์ใช้ขั้นตอนวิธีอาณานิคมผึ้งเทียม

สามารถอธิบายการทำงานของการประยุกต์ใช้ขั้นตอนวิธีอาณานิคมผึ้งเทียมควบคู่กับวิธีการทางฮิวริสติก (HABC) เพื่อใช้ในการจัดตารางงานเข้าทำงานในเครื่องคอมพิวเตอร์เสมือน ได้ดังนี้

1. กำหนดผึ้งสำรวจจำนวน n ตัว และกำหนดค่าที่เหมาะสม (Fitness) เริ่มต้นให้กับแต่ละเครื่องคอมพิวเตอร์เสมือน

กำหนดจำนวนผึ้งสำรวจ (Scout Bees) จำนวน n ตัว เพื่อใช้ในหาเครื่องคอมพิวเตอร์เสมือนที่มีความเหมาะสม และกำหนดค่าที่เหมาะสม เริ่มต้นให้กับแต่ละเครื่องคอมพิวเตอร์เสมือน ค่าที่เหมาะสมสามารถคำนวณได้จากสมการที่ (3.17)

$$Fitness_j = pe\_num_j \times pe\_mips_j + vm\_bw_j \quad (3.17)$$

เมื่อ $Fitness_j$ คือ ค่าความเหมาะสมเริ่มต้นของ VM ลำดับที่ j

pe_num_j คือ จำนวนโปรเซสเซอร์ใน VM ลำดับที่ j

pe_mips_j คือ ความเร็วในการทำงานของโปรเซสเซอร์ (Million instructions per second) ในเครื่องคอมพิวเตอร์เสมือนลำดับที่ j

vm_bw_j คือ ปริมาณแบนด์วิธของ VM ลำดับที่ j

2. ส่งผึ้งงานแบบสุ่มเข้าไปยัง VM และคำนวณค่าที่เหมาะสม

ส่งผึ้งงาน (Employed bee) จำนวน n ตัว กระจายตัวแบบสุ่ม เข้าไปในระบบเพื่อเลือกเครื่องคอมพิวเตอร์เสมือนที่เหมาะสม และคำนวนค่าที่เหมาะสม ดังสมการที่ (3.18)

$$Fitness_{ij} = \frac{\sum_{i=1}^{n} task\_length_{ij}}{Evaluate\ capacity\ of\ VM_j\ (capacity_j)} \quad (3.18)$$

เมื่อ $Fitness_{ij}$ คือ ค่าที่เหมาะสมของเครื่องคอมพิวเตอร์เสมือนลำดับที่ j จากผึ้งสำรวจตัวที่ i (Capacity of vm_j with bee number of i)

$\sum_{i=1}^{n} task\_length_{ij}$ คือ ผลรวมขนาดของงานทั้งหมดที่ทำงานใน VM ลำดับที่ j

$capacity_j$ คือ ความสามารถในการทำงานของ VM ลำดับที่ j สามารถคำนวณได้จากสมการที่ (3.19)

$$capacity_j = pe\_num_j \times pe\_mips_j + vm\_bw_j \quad (3.19)$$

เมื่อ pe_num_j คือ จำนวนโปรเซสเซอร์ใน VM ลำดับที่ j

pe_{mips_j} คือ ความเร็วในการทำงานของโปรเซสเซอร์ในเครื่องคอมพิวเตอร์เสมือนลำดับที่ j

vm_bw_j คือ ปริมาณแบนด์วิธของ VM ลำดับที่ j

3. เลือก VM ที่มีค่าเหมาะสมที่ดีที่สุดจำนวน m เครื่อง

ผึ้งเฝ้าดู (Onlooker bee) จะเลือกแหล่งอาหารที่มีค่าที่เหมาะสมดีที่สุดจำนวน M ตัว ซึ่ง m ที่ถูกเลือกจะเรียกว่าผึ้งงาน และพื้นที่ที่ผึ้งงานไปเยี่ยมชมมา จะเรียกเป็นพื้นที่ที่ถูกเลือก (Neighborhood search)

4. ส่งผึ้งจำนวนมากเข้าค้นหาในพื้นที่ที่ถูกเลือก

กำหนดผึ้งจำนวนมากเข้าค้นหาในพื้นที่ที่ถูกเลือก ส่งผึ้งจำนวนมากกว่า e ที่เลือก จำนวนผึ้งที่ส่ง nep ตัว และคำนวณค่าที่เหมาะสม ดังสมการที่ (3.20)

$$fit_{ij} = \frac{\sum_{i=1}^{n} task\_length_i + InputFile\_length}{Evaluate\ capacity\ of\ VM_j\ (capacity_j)} \quad (3.20)$$

เมื่อ $InputFile\_length$ คือ ขนาดของงานที่ต้องการเข้าใช้งาน VM

5. เลือก VM ที่ดีที่สุด

เลือก VM ที่มีค่าความเหมาะสมดีที่สุด 1 VM และส่งงาน 1 งาน เพื่อเข้าทำงานใน VM ที่ถูกเลือก

6. กำหนดงาน 1 งาน เพื่อเข้าทำงานใน VM ที่ถูกเลือก

ส่งงาน 1 งาน เพื่อเข้าทำงานใน VM ที่ถูกเลือก

7. คำนวณเพื่อดูการกระจายงานภายในกลุ่ม (Load balance)

หลังจากมีการจัดการงานให้เข้าทำงานในเครื่องคอมพิวเตอร์เสมือน จะมีการคำนวณภาระงานทั้งหมดในปัจจุบันของเครื่องคอมพิวเตอร์เสมือนทั้งหมด โดยคำนวณจากค่าส่วนเบี่ยงเบนมาตรฐาน (Standard deviation: S.D.) เพื่อวัดภาระงานทั้งหมดในเครื่องคอมพิวเตอร์เสมือน สามารถคำนวณได้ดังสมการที่ (3.21) ถึง (3.23)

$$S.D. = \sqrt{\frac{1}{n}\sum_{j=0}^{n}(X_j - \bar{X})^2} \quad (3.21)$$

เวลารวมในการทำงานของแต่ละ VM :

$$X_j = \frac{\sum_{i=1}^{k} task\_length_i}{capacity_j} \quad (3.22)$$

ค่าเฉลี่ยของเวลารวมในการทำงานของ VM ทั้งหมดในระบบ :

$$\bar{X} = \frac{\sum_{j=1}^{n} X_j}{n} \quad (3.23)$$

เมื่อ $S.D.$ คือ ค่าส่วนเบี่ยงเบนมาตรฐาน

n คือ จำนวนงานทั้งหมดในระบบ

X_j คือ เวลารวมในการทำงานของแต่ละ VM สามารถคำนวณได้จากสมการที่ (3.22)

$\bar{X}$ คือ ค่าเฉลี่ยของเวลารวมในการทำงานของ VM ทั้งหมดในระบบสามารถคำนวณได้จากสมการที่ (3.23)

ถ้า S.D. ของภาระงานของ VM มีค่าน้อยกว่าหรือเท่ากับค่าเฉลี่ยแล้วถือได้ว่า ระบบอยู่ในสภาวะสมดุล และในทางกลับกับ ถ้า S.D. มีค่ามากกว่าค่าเฉลี่ยแล้วจะถือว่า ระบบอยู่ในสภาวะของการทำงานแบบไม่สมดุล

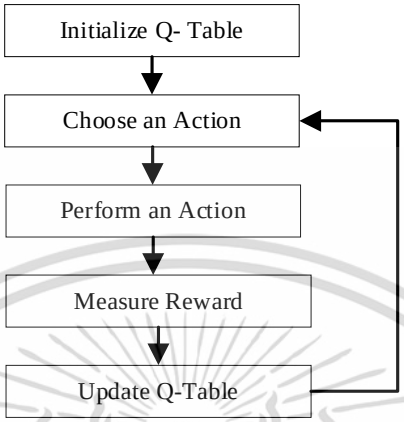
3.3 การประยุกต์ใช้ขั้นตอนอาณานิคมผึ้งเทียมร่วมกับการเรียนรู้แบบเสริมแรง

3.3.1 ขั้นตอนวิธีการเรียนรู้แบบคิว (Q-learning algorithm)

ขั้นตอนวิธีการเรียนรู้แบบเสริมกำลังที่ถูกใช้บ่อยครั้ง ได้แก่ Monte Carlo Q-learning SARSA และ DQN เป็นต้น ซึ่งอัลกอริทึมที่ถูกใช้ในงานวิจัยนี้คือ ขั้นตอนวิธีการเรียนรู้แบบคิว (Q-learning) เนื่องจากเป็นการสำรวจเพื่อค้นหาจุดหมายปลายทางที่ต้องการ เพื่อให้เอเจนต์ปรับตัวเข้ากับสภาพแวดล้อมได้

ขั้นตอนวิธีการเรียนรู้แบบคิว [10] เป็นหนึ่งในวิธีการเรียนรู้แบบเสริมกำลัง (Reinforcement learning: RL algorithm) ซึ่งวิธีการเรียนรู้แบบเสริมกำลังถือเป็นวิธีการหนึ่งในการเรียนรู้ของเครื่อง ซึ่งอนุญาตให้ตัวกระทำการตัดสินใจของระบบ (Agent) สามารถเรียนรู้ในสภาพแวดล้อมและการกระทำ (Action) โดยการเปลี่ยนสถานะ (State) เพื่อรับผลรางวัล (Reward) หรือรับบทลงโทษตามข้อเสนอแนะที่ได้รับจากการศึกษาสภาพแวดล้อม (Environment) โดยที่จุดประสงค์หลักของวิธีการเรียนรู้แบบเสริมกำลัง คือ การเรียนรู้ของตัวกระทำที่เกิดจากการลองผิดลองถูกระหว่างตัวกระทำกับสภาพแวดล้อม โดยตัวกระทำจะสามารถรับสถานการณ์ของสภาพแวดล้อม ผ่านสถานะและเลือกการกระทำที่ส่งผลต่อสภาพแวดล้อม โดยหวังว่าจะได้รับรางวัลที่ดีที่สุด รวมทั้งเรียนรู้ผ่านข้อผิดพลาดในอดีตที่เกิดขึ้นกระบวนการตัดสินใจของมาร์คอฟ (Markov decision process: MDP) เป็นเฟรมเวิร์คสำหรับการตัดสินใจ (Decision making) ของตัวกระทำ

เนื่องจากสภาพแวดล้อมมีความไม่แน่นอน ทำให้ผลลัพธ์ที่ได้ในบางครั้งไม่แน่นอน (Stochastic) กล่าวคือ การที่ตัวกระทำเลือกทำการกระทำเดิม สำหรับสถานการณ์ (State) เดิม อาจจะไม่ได้ผลลัพธ์แบบเดิมเสมอไป กระบวนการขั้นตอนวิธีการเรียนรู้แบบคิวแสดงได้ดังรูปที่ 3.2



รูปที่ 3.2 กระบวนการขั้นตอนวิธีการเรียนรู้แบบคิว

กำหนดให้เซตของสถานะ $S = \{s_1, s_2, s_3, \dots, s_n\}$ ในสภาพแวดล้อมและแต่ละสถานะมีเซตของการกระทำ $A = \{a_1, a_2, a_3, \dots, a_m\}$ ตัวกระทำเลือกการกระทำ $a_t \epsilon A$ ณ เวลา t ในสถานะ $s_t \epsilon S$ เพื่อส่งต่อไปยังสถานะถัดไป $s_{t+1} \epsilon S$ ผ่านกระบวนการเปลี่ยนผ่านและรับรางวัล r_{t+1} จากสภาพแวดล้อม ในการทำงานจำเป็นที่จะต้องเลือกการกระทำในการดำเนินการที่เหมาะสม เพื่อปรับปรุงค่า Q-value ของแต่ละสถานะให้มากที่สุด ซึ่งถือเป็นวัตถุประสงค์หลักของค้นหานโยบายที่เหมาะสมที่สุดในการประมวลผลแบบกลุ่มเมฆ ค่าของ Q-value function นั้นขึ้นอยู่กับการเลือกของการกระทำในสถานะกำหนดให้ตัวกระทำในสถานะ s_t และเลือกการกระทำ a_t ค่า Q-value function นั้นคาดว่าจะย้ายไปยังสถานะที่ดีที่สุดและมีการเพิ่มผลตอบแทนที่คาดหวังให้มีค่าสูงที่สุด (Maximize the total expected reward) ในสภาพแวดล้อม ค่า Q-value สามารถคำนวณได้โดยใช้สมการที่ 3.24

$$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha[r_t + \gamma max_{a+1} Q(s_{t+1}, a_{t+1})] \qquad (3.24)$$

โดยที่ α คือ อัตราการเรียนรู้ (Learning rate)

$\gamma(0 < \gamma < 1)$ คือ ปัจจัยส่วนลด (Discount factor) และผลกระทบต่อสถานะโดยการกระทำก่อนหน้า

r_t คือ บทลงโทษหรือรางวัลที่ได้รับจากการดำเนินการในสถานะ s_t

ค่า Q-value ได้จากการสร้าง Q-table ซึ่งเก็บสถานะที่เป็นไปได้ทั้งหมดและ Q-value และการกระทำที่เหมาะสม โดยที่ขั้นตอนวิธีการเรียนรู้แบบคิวพยายามสร้างสถานะที่เหมาะสมที่สุดจากประสบการณ์และมีการนำขั้นตอนวิธีแบบละโมบ (Greedy algorithm) เข้ามาใช้ในขั้นตอนวิธีการเรียนรู้แบบคิว สามารถคำนวณ Q-value ได้จากสมการที่ (3.25)

$$Q_{t+1}(s_t,a_t) = Q(s_t, a_t) + \alpha\left[r_t + \gamma max_{\acute{a}_t}[Q_t(\delta(s_t,a_t),\acute{a}_t)] - Q_t(s_t,a_t)\right] \quad (3.25)$$

โดยที่ α คือ อัตราการเรียนรู้ซึ่งคำนวณได้จาก α =1/(1+จำนวนครั้งทั้งหมดที่ไปยังสถานะ s_t)

δ คือ ฟังก์ชันการเปลี่ยนสถานะ (Transition function)

$\max_a Q(s_{t+1},a)$ คือ ประมาณการค่าในอนาคตที่เหมาะสม ซึ่ง action ที่เป็นไปได้ในการประมวลผลแบบกลุ่มเมฆที่เกี่ยวกับโหลดบาลานซ์ (Load balancing) คือ การจัดสรรทรัพยากรในเครื่องคอมพิวเตอร์เสมือน ขั้นตอนวิธีการเรียนรู้แบบคิวสามารถสรุปได้ดัง Algorithm 1

Algorithm 1: Q-update(S)

1. Set values for learning rate α, discount factor γ, reward r
2. Initialize Q-values
3. For $i = 1$ to n # n is the number of states
4. For $j = 1$ to p # p is the number of actions in each state
5. $Q(s_i,a_j) = 0$
6. End for
7. End for
8. Select an action a_i from $A = \{a_1,a_2,a_3,\dots,a_m\}$ and execute it and go to next state s_{t+1}
9. Calculate the learning rate
10. Calculate the reward
11. Update Q_{t+1} by (3.25)
12. Repeat this step for new state until it converges

3.3.2 การประยุกต์ใช้ขั้นตอนอาณานิคมผึ้งเทียม

การประยุกต์ใช้ขั้นตอนอาณานิคมผึ้งเทียม [6, 8] จัดเป็นขั้นตอนวิธีการเมตาฮิวริสติกประเภทความฉลาดแบบกลุ่ม ซึ่งระบบกลุ่มประกอบด้วยตัวกระทำการตัดสินใจของระบบ (Agent) มีการสื่อสารระหว่างตัวกระทำการตัดสินใจของระบบ และสภาพแวดล้อม โดยในกลุ่มฝูงเดียวกันจะมีเป้าหมายที่เหมือนกัน ซึ่งใน การทำงานของขั้นตอนอาณานิคมผึ้งเทียม ผึ้งจะมีเป้าหมายเดียวกัน คือ เพื่อค้นหาการหาแหล่งอาหารที่ดีที่สุด โดยแหล่งอาหาร (Food source) แสดงถึง ชุดของคำตอบที่เป็นไปได้ทั้งหมด และแต่ละตัวกระทำถูกแทนด้วยผึ้ง

ในขั้นตอนอาณานิคมผึ้งเทียม ตัวกระทำการตัดสินใจถูกแบ่งประเภทตามหน้าที่ของพวกผึ้ง ซึ่งสามารถแบ่งออกได้เป็น 3 กลุ่ม ได้แก่ ผึ้งงาน (Employed bees) ผึ้งเฝ้าดู (Onlooker bees) และผึ้งสำรวจ (Scout bees) ผึ้งแต่ละกลุ่มมีพฤติกรรมแบบ Explorative และ Exploitative ที่แตกต่างกัน คือ Explorative คือ พฤติกรรมที่เกี่ยวข้อกับการค้นหาแหล่งอาหารในพื้นที่การค้นหา (Search space) เพื่อหลีกเลี่ยงการค้นหาที่แหล่งอาหารในท้องถิ่น (Local) ในทางตรงกันข้าม Exploitative คือพฤติกรรมการแสวงหาผลประโยชน์ที่เกี่ยวข้องกับการค้นหาแหล่งอาหารที่ดีกว่าในบริเวณที่ใกล้กับแหล่งอาหารปัจจุบัน โดยในงานวิจัยนี้ ได้ใช้ขั้นตอนวิธีการเรียนรู้แบบคิวเพื่อปรับปรุงขั้นตอนในการหาคำตอบของปัญหาให้มีความเหมาะสมมากขึ้น จากการศึกษาพบว่าขั้นตอนอาณานิคมผึ้งเทียม (ABC algorithm) มีความสามารถในพฤติกรรมการหาแหล่งอาหารแบบ Explorative แต่อ่อนแอในด้านพฤติกรรมแบบ Exploitative ดังนั้นจึงตั้งเป้าที่จะปรับปรุงพฤติกรรมแบบ Exploitative ในส่วนของผึ้งเฝ้าดู ขั้นตอนการทำงานสามารถอธิบายได้ดังนี้

ในขั้นตอนแรกของขั้นตอนอาณานิคมผึ้งเทียม แทนตำแหน่งของแหล่งอาหารด้วยคำตอบที่เป็นไปได้ของปัญหา เริ่มต้นแหล่งอาหารจะถูกสร้างขึ้นแบบสุ่ม ดังแสดงในสมการที่ (3.26) และหลังจากนั้นผึ้งงานจะทำการสำรวจหาแหล่งอาหาร (VM) ในการทำงานกำหนดค่าเริ่มต้นของ Q-table มีค่าเท่ากับ 0

$$x_{i,j}^{0} = x_j^{min} + rand(0,1) * (x_j^{max} - x_j^{min}) \tag{3.26}$$

โดยที่ x_j^{min} คือ ขอบล่างของ j^{th} พารามิเตอร์การเพิ่มประสิทธิภาพ (Optimization parameter)

x_j^{max} คือ ขอบบนของ j^{th} พารามิเตอร์การเพิ่มประสิทธิภาพ

ในขั้นตอนอาณานิคมผึ้งเทียมแบ่งการทำงานย่อยของอัลกอริทึมออกเป็น 3 ส่วน ได้แก่ ส่วนของผึ้งงาน ผึ้งเฝ้าดู และส่วนของผึ้งสำรวจ โดยจะทำงานซ้ำทั้ง 3 ขั้นตอนจนกว่าจะครบเงื่อนไขที่กำหนดหรือจำนวนรอบสูงสุดที่กำหนด

ส่วนของผึ้งงาน ทำการค้นหาตำแหน่งของแหล่งอาหาร ($v_{i,j}^{t}$) ของแหล่งอาหารในปัจจุบัน ($\bar{x}_i^t$) โดยใช้สมการที่ (3.27)

$$v_{i,j}^{t} = x_{i,j}^{t} + rand[-1,1] * (x_{i,j}^{t} - x_{k,j}^{t}) \quad (3.27)$$

โดยที่ $v_{i,j}^{t}$ คือ j^{th} พารามิเตอร์การเพิ่มประสิทธิภาพของ $\bar{v}_i^t$

k คือ ลำดับของแหล่งอาหาร

ถ้าแหล่งอาหารใหม่ $\bar{v}_i^t$ ให้ค่าความเหมาะสมที่ดีกว่าแหล่งอาหารปัจจุบัน x_i^t โดยที่ $Fit(\bar{v}_i^t) > Fit(\bar{x}_i^t)$ ผึ้งงานจะทิ้งแหล่งอาหารเดิมและจดจำตำแหน่งใหม่ ซึ่งในขั้นตอนนี้จะมีการปรับปรุงค่าใน Q-table (Reward-penalty scheme) โดยใช้สมการที่ (3.25) หากแหล่งอาหารใหม่ให้ค่าความเหมาะสมที่ดีกว่า ผึ้งงานไม่เพียงแต่จะแทนที่แหล่งอาหารเดิมด้วยแหล่งอาหารใหม่ แต่ยังมีการปรับปรุงค่า Q-value โดยการให้รางวัลกับแหล่งอาหารใหม่ที่เลือก และทำการลงโทษกับแหล่งอาหารเดิม

ในทางกลับกัน หากแหล่งอาหารเดิมมีค่าความเหมาะสมดีกว่าแหล่งอาหารใหม่ แหล่งอาหารใหม่จะถูกลงโทษ และให้รางวัลกับแหล่งอาหารเดิม Q-table จะมีการปรับปรุงค่าทุกครั้งที่ผึ้งงานสามารถค้นหาแหล่งอาหารที่เหมาะสม ดังนั้น ถ้า EB คือจำนวนของผึ้งงาน จำนวนรอบในการปรับปรุงค่าใน Q-Table เท่ากับ EB ครั้ง

ในส่วนของผึ้งเฝ้าดูนั้นจะทำการเลือกแหล่งอาหารของผึ้งงาน โดยทำการเลือกจากค่า Q-value ในตาราง Q-table ในทางตรงกันข้าม ผึ้งเฝ้าดูจะทำการค้นหาแหล่งอาหารแหล่งใหม่ โดยใช้สมการที่ (3.28) และทำการแทนที่แหล่งอาหารเดิมด้วยแหล่งอาหารใหม่ หากแหล่งอาหารใหม่มีค่าความเหมาะสมมากกว่า อีกทั้งยังทำการปรับปรุงค่า Q-value

$$v_{i,j}^{t} = x_{i,k}^{t} + \emptyset_{i,j}^{t} \cdot r_j^t \cdot (x_{i,k}^{t} - x_{BSF,k}^{t}) \quad (3.28)$$

โดยที่ $v_{i,j}^{t}$ คือ พารามิเตอร์การเพิ่มประสิทธิภาพของแหล่งอาหารที่ $\bar{v}_i^t$

$\emptyset \in [-1,1]$ คือ มีค่าระหว่าง -1 ถึง 1

$x_{BSF,k}^{t}$ คือ พารามิเตอร์การเพิ่มประสิทธิภาพของแหล่งอาหารที่ดีที่สุดซึ่งเกิดจากการสุ่ม

ส่วนของผึ้งสำรวจ หากครบจำนวนรอบในการค้นหาแหล่งอาหารแล้วไม่พบแหล่งอาหารที่ดีกว่า แหล่งอาหารนั้นจะถูกทิ้ง และผึ้งสำรวจจะทำการสุ่มเพื่อค้นหาแหล่งอาหารแหล่งใหม่

ขั้นตอนการทำงานของขั้นตอนวิธี MOABCQ ถูกนำเสนอใน Algorithm2 ดังนี้

Algorithm 2. MOABCQ algorithm

Initialization:

1. Initialize the population and calculate individual fitness values
2. Set up the parameter: best solution, maximum number of iterations, the population size
3. Find the best solution
4. while stopping criteria satisfied

Employed Bees Phase:

5. For each position do
6. Update position of employed bee by (3.27)
7. Estimate the new position
8. If fitness value of new position is better
9 Replace the current position with the new position
10 End if
11. Calculate probability and update the Q-table for select position in the onlooker bee phase
12. End for

Onlooker Bees Phase:

13. For each onlooker do
14. Select a position based on Q-value and probability
15. Update position of onlooker bee by (3.28)
16. Estimate the new position
17. If fitness value of new position is better
18. Replace the current position with the new position
19. End if
20. update the Q-table using (3.25)
21. End for
22. Find the best solution (Q-table)

Scout Bees Phase:

23. For each position do
24. Abandon the solution that have not been updated and generate new solutions randomly
25. update the Q-table using (3.25)
26. End for
27. End while

บทที่ 4

การทดลองและผลการทดลอง

จากบทที่ 3 ได้กล่าวถึงการประยุกต์ใช้ปัญญาประดิษฐ์สำหรับปัญหาการจัดตารางงานที่มีหลายวัตถุประสงค์ (Multi-objective optimization problem) ในการประมวลผลแบบกลุ่มเมฆในกรณีที่เครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในแตกต่างกัน โดยใช้ขั้นตอนอาณานิคมผึ้งเทียมและวิธีการเรียนรู้แบบเสริมแรง โดยใช้ขั้นตอนวิธีการเรียนรู้แบบคิว (Q-learning) ซึ่งอัลกอริทึมที่นำเสนอนี้เรียกว่า “MOABCQ” อีกทั้งมีการพิจารณาลำดับการเข้าทำงานให้เหมาะสมกับสภาพแวดล้อมของทรัพยากรที่มีอยู่ เพื่อหารูปแบบการจัดตารางงานที่เหมาะสมที่สุด โดยบทนี้อธิบายการตั้งค่าพารามิเตอร์และรายละเอียดในการทดลองเพื่อประเมินประสิทธิภาพของวิธีการที่นำเสนอ (MOABCQ) ในการทดลองครั้งนี้ได้ทำการเปรียบเทียบกับวิธีการจัดตารางงานแบบฮิวริสติกที่เป็นที่รู้จัก เช่น ขั้นตอนการจัดตารางงานโดยใช้ Max-Min [53] ขั้นตอนวิธีการจัดตารางงานแบบมาก่อนได้ก่อน (First Come First Serve algorithm: FCFS algorithm) ขั้นตอนการจัดตารางงานแบบงานขนาดใหญ่ที่สุดทำก่อน (Largest Job First algorithm: LJF algorithm) และยิ่งไปกว่านั้น ยังได้ทำการเปรียบเทียบการจัดตารางงานโดยใช้วิธีการทางเมตาฮิวริสติกที่เป็นที่นิยม ได้แก่ ขั้นตอนวิธีการหาคำตอบเหมาะสมที่สุดแบบกลุ่มอนุภาค (Particle Swarm Optimization: PSO algorithm) ขั้นตอนวิธีการหาค่าเหมาะสมที่สุดแบบนกกาเหว่า (Cuckoo Search: CS algorithm) ขั้นตอนวิธีอาณานิคมมด (Ant Colony Optimization: ACO algorithm) และขั้นตอนวิธีอาณานิคมผึ้งเทียม (ABC algorithm)

4.1 ข้อมูลที่ใช้ในการทดสอบประสิทธิภาพ

ข้อมูลที่ใช้ในการทดสอบประสิทธิภาพของวิธีการที่นำเสนอ ใช้ชุดข้อมูลดังแสดงในตารางที่ 4.1 ในการทดสอบประสิทธิภาพ โดยมีการแบ่งชุดข้อมูลออกเป็น 12 ชุดข้อมูล สามารถแบ่งออกได้เป็น 4 ประเภทดังนี้ ประเภทที่ 1 ชุด D_1 – D_3 คือ ชุดข้อมูลแบบสุ่มอิสระที่ไม่มีข้อกำหนดของรูปแบบที่แน่นอน ประเภทที่ 2 ชุด D_4 – D_6 คือชุดข้อมูลแบบสุ่มที่มีการกระจายของข้อมูลแบบแจกแจงปกติ (Normal distribution) ประเภทที่ 3 ชุด D_7 – D_9 คือ ชุดข้อมูลแบบสุ่มที่มีการกระจายข้อมูลแบบเบ้ขวา (Right-skewed distribution) และประเภทที่ 4 ชุด D_{10} – D_{12} คือชุดข้อมูลแบบสุ่มที่มีการกระจายข้อมูลแบบเบ้ซ้าย (Left-skewed distribution) ชุดข้อมูลแต่ละประเภทมีช่วงขนาดของงานที่แตกต่างกัน และชุดข้อมูลแต่ละชุดมีการจัดจำนวนงานออกเป็นชุด ๆ ละ 100, 200, 300, ..., และ 1500 งานตามลำดับ

ตารางที่ 4.1 ชุดข้อมูลที่ใช้ในการทดสอบประสิทธิภาพ

ประเภทของชุดข้อมูล	ชุดข้อมูล	ขนาดของงาน (MI[a])
สุ่มอิสระ	D_1	[5000, 20000]
	D_2	[15000, 35000]
	D_3	[25000, 45000]
การกระจายข้อมูลแบบแจกแจงปกติ	D_4	[5000, 20000]
	D_5	[15000, 35000]
	D_6	[25000, 45000]
การกระจายข้อมูลแบบเบ้ขวา	D_7	[5000, 20000]
	D_8	[15000, 35000]
	D_9	[25000, 45000]
การกระจายข้อมูลแบบเบ้ซ้าย	D_{10}	[5000, 20000]
	D_{11}	[15000, 35000]
	D_{12}	[25000, 45000]

หมายเหตุ: [a]MI คือ Million instruction

4.2 การเตรียมข้อมูล

ในขั้นแรกของการทดสอบประสิทธิภาพจะทำการจัดเรียงข้อมูล ดังตารางที่ 4.2 โดยแบ่งการจัดเรียงข้อมูลเป็น 3 รูปแบบ คือ

1. จัดเรียงข้อมูลตามลำดับการมาถึงของงาน (FCFS)
2. จัดเรียงข้อมูลตามขนาดของงาน โดยเรียงข้อมูลจากเล็กที่สุดไปหาขนาดใหญ่ที่สุด (SJF)
3. จัดเรียงข้อมูลตามขนาดของงาน โดยเรียงข้อมูลจากใหญ่ที่สุดไปหาขนาดเล็กที่สุด (LJF)

ตารางที่ 4.2 ตัวอย่างในการจัดเรียงข้อมูล

FCFS	SJF	LJF
5163	5163	11965
7832	5322	11290
11290	7256	10897
10897	7832	9159
7256	9159	7832
11965	10897	7256

4.3 เครื่องมือที่ใช้ในงานวิจัย

เครื่องคอมพิวเตอร์ที่ใช้ในการทดสอบ มีคุณสมบัติ ดังนี้

Notebook: Dell G7 7590

หน่วยประมวลผลกลาง (CPU) : Intel® Core (TM) i7-8750H Processor 2.20 GHz.

หน่วยความจำหลัก (RAM) : 16.0 GB GDDR15

หน่วยความจำสำรอง (Hard Disk) : 1TB

ระบบปฏิบัติการ (OS) : Windows 10 Pro

โปรแกรมที่ใช้ในการจำลองระบบการประมวลผลแบบกลุ่มเมฆ: CloudSim-3.0.3

4.4 ผลการทดลองเพื่อหาพารามิเตอร์ที่เหมาะสมของขั้นตอนอาณานิคมผึ้งเทียม

4.4.1 การตั้งค่าพารามิเตอร์สภาพแวดล้อมในการทดลอง

การตั้งค่าพารามิเตอร์แบบจำลองเพื่อประเมินประสิทธิภาพของระบบการประมวลผลแบบกลุ่มเมฆ กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในที่แตกต่างกัน แสดงค่าดังตารางที่ 4.3

ตารางที่ 4.3 ค่าพารามิเตอร์ของแบบจำลองการประมวลผลแบบกลุ่มเมฆ กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมแตกต่างกันทั้งหมด

Type	Parameter	Value
Datacenter	Number of Datacenter	10
	Number of Host	5
	Type of Manager	Space_Shared, Time_Shared
	Number of PE per Host	4
	MIPS of PE	2000
	Host Memory (RAM) (MB)	10240
	Datacenter Cost	10
Virtual Machines (VMs)	Total number of VM	60
	MIPS of PE	1000 - 5000
	Number of PE per VM	2
	VM Memory (RAM) (MB)	512 - 4096
	Bandwidth (Bit)	1000 - 10000
	Type of Manager	Time_Shared

ตารางที่ 4.3 ค่าพารามิเตอร์ของแบบจำลองการประมวลผลแบบกลุ่มเมฆ กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมแตกต่างกัน (ต่อ)

Type	Parameter	Value
Task	Total number of Task	100 – 1500
	Length of Task (MI)	5000 - 20000
	Number of PE per requirement	1
	Type of Manager	Space_Shared

หมายเหตุ: PE = Processing element, MIPS = Million instructions per second, MB = Megabyte, MI = Million instruction

4.4.2 การตั้งค่าพารามิเตอร์ในขั้นตอนวิธีอาณานิคมผึ้งเทียม

ในการทดลอง ได้มีแบ่งการกำหนดค่าพารามิเตอร์ออกเป็น 6 กรณี ดังนี้ 60 (เท่ากับจำนวนเครื่องคอมพิวเตอร์เสมือน), 120, 240, 480 , 960 และ 1920 แต่ในที่นี้แสดงให้เห็นเพียง 3 กรณีที่ปรากฏถึงผลลัพธ์ที่ชัดเจน รายละเอียดของการตั้งค่าพารามิเตอร์สำหรับขั้นตอนวิธีอาณานิคมผึ้งเทียม (ABC) แสดงค่าดังตารางที่ 4.4

ตารางที่ 4.4 ค่าพารามิเตอร์ของขั้นตอนวิธีอาณานิคมผึ้งเทียม

Symbol	Parameter	Case I	Case II	Case III
n	Number of scout bees	60	240	960
m	Number of sites selected out of n visited sites	6	24	96
e	Number of best sites out of m selected sites	1	1	1
nep	Number of bees recruited for best e sites	48	192	768
nsp	Number of bees recruited for other (m-e) selected sites	12	48	192

4.4.3 ผลการทดสอบประสิทธิภาพ เมื่อจำนวนผึ้งในระบบมีการเปลี่ยนแปลง

การประเมินประสิทธิภาพในการจัดตารางงานในเครื่องคอมพิวเตอร์เสมือน โดยใช้ขั้นตอนวิธีอาณานิคมผึ้งเทียมควบคู่กับการจัดลำดับงานแบบฮิวริสติก (HABC_FCFS, HABC_SJF และ HABC_LJF) จะทำการเปรียบเทียบกับการจัดลำดับงานแบบฮิวริสติกเพียงอย่างเดียว (FCFS, SJF และ LJF) โดยทำการทดลองกับชุดข้อมูล 2 รูปแบบ คือ ชุดข้อมูลการสุ่มอิสระแบบไม่มีข้อกำหนด (ไม่คำนึงถึงรูปแบบการกระจายของข้อมูล) และชุดข้อมูลการสุ่มที่มีรูปแบบการกระจายแบบแจกแจงปกติ ในการทดลองใช้ชุดข้อมูลแสดงดังตารางที่ 4.5 เพื่อทำการเปรียบเทียบความสามารถของการ

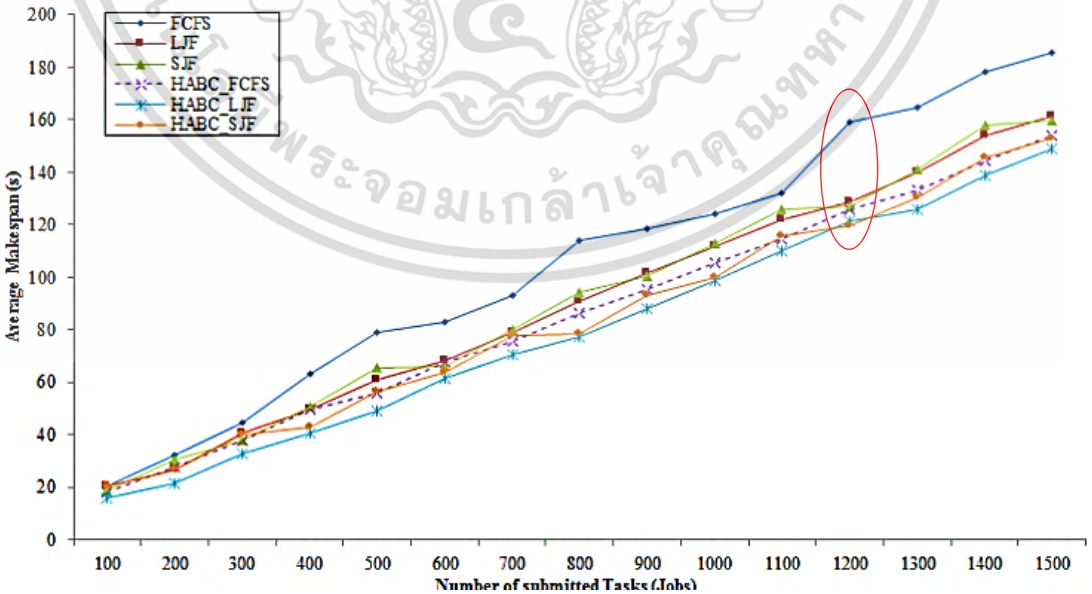
ทำงานของทั้ง 2 ชุดข้อมูลกับจำนวนงานที่มีการเปลี่ยนแปลงไป ซึ่งในการทดสอบกำหนดให้ เครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในที่แตกต่างกัน โดยกำหนดให้จำนวนของเครื่องเสมือนมีค่าคงที่ตลอดการทดลอง แต่จำนวนงานเพิ่มขึ้นดังนี้ 100, 200, 300, ..., 1500 งาน กำหนดจำนวนรอบในการทดลองมีค่าเท่ากับ 20 รอบ จากนั้นกำหนดให้แนวแกน X แทนปริมาณงานที่มีการเปลี่ยนแปลงในระบบ และแนวแกน Y แทนเวลาเฉลี่ยรวมของการทำงานในระบบ โดยแบ่งการทดสอบประสิทธิภาพออกเป็น 2 กรณี คือ

ตารางที่ 4.5 ชุดข้อมูลที่ใช้ในการทดสอบประสิทธิภาพของการทดสอบกรณีที่ 4.4.3

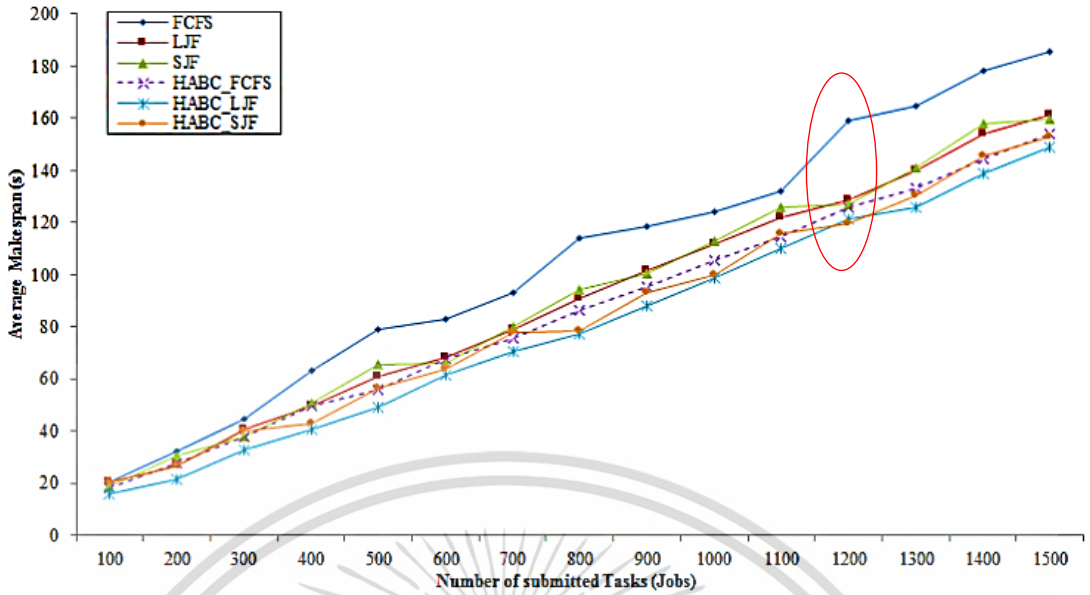
ประเภทของชุดข้อมูล	ชุดข้อมูล	ขนาดของงาน (MI)
สุ่มอิสระ	D_1	[5000, 20000]
การกระจายข้อมูลแบบแจกแจงปกติ	D_4	[5000, 20000]

กรณีที่ 1: เปรียบเทียบเวลารวมเฉลี่ยในการทำงานโดยใช้ชุดข้อมูลประเภทแบบสุ่มอิสระ

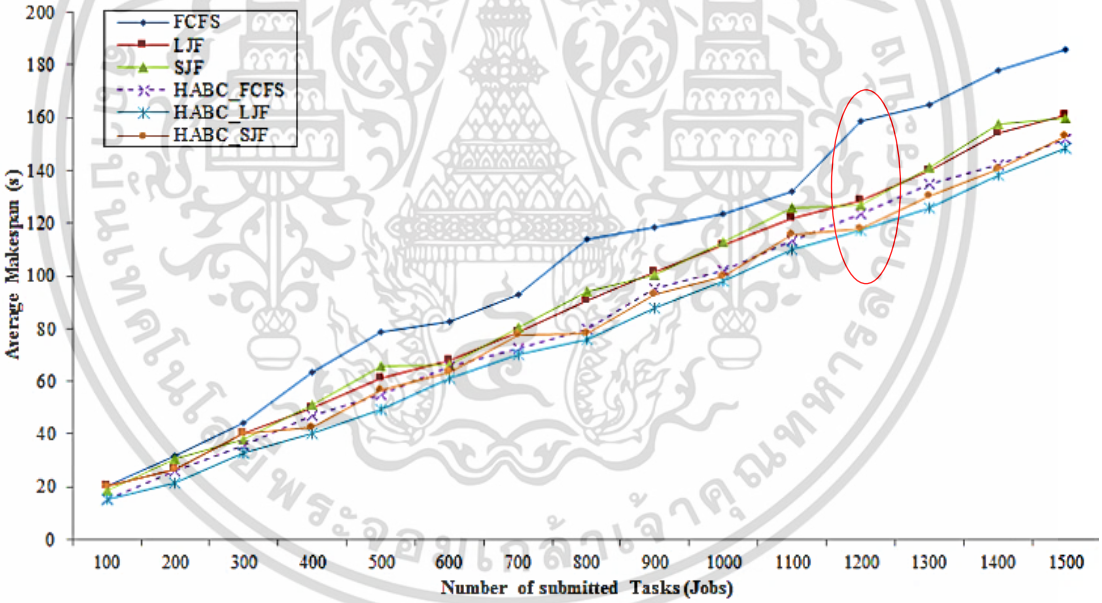
ในการเปรียบเทียบเวลารวมเฉลี่ยในการทำงานกับจำนวนงานที่มีการเปลี่ยนแปลงในระบบกับชุดข้อมูลสุ่มแบบอิสระ ซึ่งใช้ชุดข้อมูล D_1 เพื่อประเมินประสิทธิภาพของวิธีการที่นำเสนอ ซึ่งแบ่งออกเป็น 3 วิธี คือ HABC_FCFS, HABC_SJF และ HABC_LJF โดยทำการเปรียบเทียบกับการจัดตารางงานโดยใช้วิธีการทางฮิวริสติก ได้แก่ FCFS, SJF และ LJF ซึ่งผลจากการทดสอบความสามารถในการทำงานโดยเปรียบเทียบกับจำนวนงานที่มีการเปลี่ยนแปลงและจำนวนผึ้งในระบบมีการเปลี่ยนแปลง แสดงได้ดังรูปที่ 4.1 – 4.3 และตารางที่ 4.6



รูปที่ 4.1 เวลาทำงานรวมเฉลี่ยของชุดข้อมูล D_1 เมื่อกำหนดค่าพารามิเตอร์ของ ABC โดยใช้ Case I



รูปที่ 4.2 เวลาทำงานรวมเฉลี่ยของชุดข้อมูล D_1 เมื่อกำหนดค่าพารามิเตอร์ของ ABC โดยใช้ Case II



รูปที่ 4.3 เวลาทำงานรวมเฉลี่ยของชุดข้อมูล D_1 เมื่อกำหนดค่าพารามิเตอร์ของ ABC โดยใช้ Case III

ตารางที่ 4.6 เวลาในการทำงานรวมโดยใช้ชุดข้อมูลประเภทแบบสุ่มอิสระ

Task	Algorithm	Processing Time (s)								
		Case I			Case II			Case III		
		min	*max*	*avg*	*min*	*max*	*avg*	*min*	*max*	*avg*
100	HABC_FCFS	**14.51**	20.49	17.94	**13.74**	18.98	16.27	**13.01**	16.39	15.38
	HABC_LJF	14.85	**17.00**	**15.85**	14.74	**15.89**	**15.72**	14.31	**15.89**	**15.31**
	HABC_SJF	19.75	20.59	20.41	16.13	20.59	20.13	14.85	20.59	20.20
200	HABC_FCFS	24.42	32.93	27.79	23.06	32.15	25.34	22.85	27.70	25.87
	HABC_LJF	**21.30**	**22.00**	**21.68**	**21.39**	**21.63**	**21.52**	**21.44**	**21.60**	**21.52**
	HABC_SJF	25.70	28.32	26.71	26.50	26.89	26.74	25.90	26.89	26.63
300	HABC_FCFS	**32.65**	44.29	37.90	**31.85**	40.90	34.95	**32.33**	36.76	35.73
	HABC_LJF	32.69	**33.18**	**32.89**	32.75	**32.95**	**32.83**	32.65	**32.91**	**32.81**
	HABC_SJF	39.20	41.10	40.33	39.79	40.94	40.57	39.72	40.78	40.26
400	HABC_FCFS	46.03	54.33	49.52	43.38	53.42	47.09	46.25	48.09	47.25
	HABC_LJF	**40.44**	**40.77**	**40.60**	**40.30**	**40.55**	**40.44**	**40.37**	**40.58**	**40.47**
	HABC_SJF	42.15	43.16	42.69	42.44	42.90	42.72	42.38	42.90	42.72
500	HABC_FCFS	53.46	60.60	56.07	50.16	57.76	53.90	52.24	56.48	54.83
	HABC_LJF	**49.06**	**49.76**	**49.36**	**48.94**	**49.24**	**49.08**	**48.96**	**49.13**	**49.07**
	HABC_SJF	56.10	57.13	56.70	55.72	56.82	56.48	55.68	56.83	56.50
600	HABC_FCFS	62.50	71.66	67.45	**59.75**	68.76	64.33	**61.14**	66.77	65.57
	HABC_LJF	**61.40**	**61.65**	**61.55**	61.25	**61.42**	**61.36**	61.23	**61.45**	**61.36**
	HABC_SJF	63.01	64.01	63.68	63.52	63.89	63.71	63.43	63.95	63.68
700	HABC_FCFS	**70.37**	83.00	75.50	**69.48**	79.25	73.35	**69.41**	74.30	72.72
	HABC_LJF	70.52	**70.95**	**70.68**	70.40	**70.61**	**70.50**	70.41	**70.61**	**70.50**
	HABC_SJF	77.14	77.95	77.68	77.15	77.93	77.56	77.20	77.71	77.54
800	HABC_FCFS	82.56	88.95	86.15	79.08	88.21	82.82	78.77	80.32	79.76
	HABC_LJF	**76.09**	81.12	**77.45**	**76.02**	**76.30**	**76.20**	**76.05**	**76.35**	**76.18**
	HABC_SJF	77.73	**78.76**	78.34	78.12	78.43	78.30	78.05	78.44	78.31
900	HABC_FCFS	91.21	100.74	95.18	91.25	96.33	94.81	89.85	100.07	94.99
	HABC_LJF	**87.79**	**88.16**	**87.99**	**87.68**	**87.96**	**87.86**	**87.68**	**87.94**	**87.87**
	HABC_SJF	92.88	93.46	93.19	93.02	93.38	93.20	92.93	93.39	93.16
1000	HABC_FCFS	100.34	109.93	105.48	98.88	106.66	103.61	98.46	103.54	101.99
	HABC_LJF	**98.31**	**98.60**	**98.47**	**98.27**	**98.50**	**98.35**	**98.24**	**98.38**	**98.32**
	HABC_SJF	99.33	100.14	99.82	99.65	100.20	99.98	99.61	100.19	99.99
1100	HABC_FCFS	111.06	119.90	114.66	110.73	121.16	113.97	110.51	113.54	113.16
	HABC_LJF	**109.97**	**110.26**	**110.12**	**109.90**	**110.15**	**110.01**	**109.77**	**110.16**	**110.02**
	HABC_SJF	114.81	115.74	115.51	115.38	115.84	115.64	115.19	115.91	115.62
1200	HABC_FCFS	121.10	131.13	125.82	119.67	128.16	124.21	122.01	124.69	123.69
	HABC_LJF	**117.07**	**121.94**	121.06	**117.05**	121.80	118.50	**117.02**	**117.24**	**117.16**
	HABC_SJF	117.48	129.82	**119.85**	117.94	**118.36**	**118.10**	117.78	118.29	118.10
1300	HABC_FCFS	129.66	137.41	133.39	129.81	137.32	133.23	128.80	136.18	134.75
	HABC_LJF	**125.76**	**126.18**	**125.95**	**125.70**	**125.98**	**125.87**	**125.73**	**125.98**	**125.88**
	HABC_SJF	129.82	130.42	130.12	129.95	130.36	130.15	129.92	130.33	130.19
1400	HABC_FCFS	139.42	152.43	144.64	139.26	145.42	141.97	139.75	143.17	142.12
	HABC_LJF	**138.49**	**138.73**	**138.58**	**138.21**	**138.62**	**138.50**	**138.19**	**138.58**	**138.48**
	HABC_SJF	144.15	146.06	145.39	138.35	145.33	141.49	138.20	145.30	140.83
1500	HABC_FCFS	149.20	158.22	154.12	148.25	158.43	151.08	149.17	153.38	152.13
	HABC_LJF	**148.68**	**148.93**	**148.79**	**147.58**	**148.86**	**148.68**	**148.65**	**148.87**	**148.74**
	HABC_SJF	152.15	152.97	152.71	152.39	153.15	152.79	152.42	153.04	152.79

จากรูปที่ 4.1-4.3 และตารางที่ 4.6 แสดงเวลาทำงานรวมเฉลี่ยโดยทำการเปรียบเทียบขั้นตอนการจัดตารางในเครื่องคอมพิวเตอร์เสมือนที่มีสภาพแวดล้อมภายในแตกต่างกัน โดยใช้ขั้นตอนวิธีอาณานิคมผึ้งเทียมควบคู่กับการจัดลำดับงานแบบฮิวริสติก (HABC) เปรียบเทียบกับการจัดลำดับงานแบบฮิวริสติกเพียงอย่างเดียว ผลจากการทดลองแสดงให้เห็นว่า หลังจากใช้อัลกอริทึม HABC สำหรับการจัดตารางงาน เวลาทำงานรวมเฉลี่ยมีค่าลดลง เมื่อเปรียบเทียบกับการจัดตารางงานโดยใช้วิธีการทางฮิวริสติก นอกจากนี้เมื่อมีการปรับเปลี่ยนพารามิเตอร์ของ ABC เพื่อค้นหาพารามิเตอร์ที่เหมาะสมที่สุด ผลจากการทดลองแสดงให้เห็นว่า เมื่อกำหนดพารามิเตอร์โดยใช้ Case III พบว่า HABC_LJF มีประสิทธิเหนือกว่าวิธีการอื่นๆ เนื่องจากให้ค่าเวลาที่ใช้ในการทำงานรวมน้อย

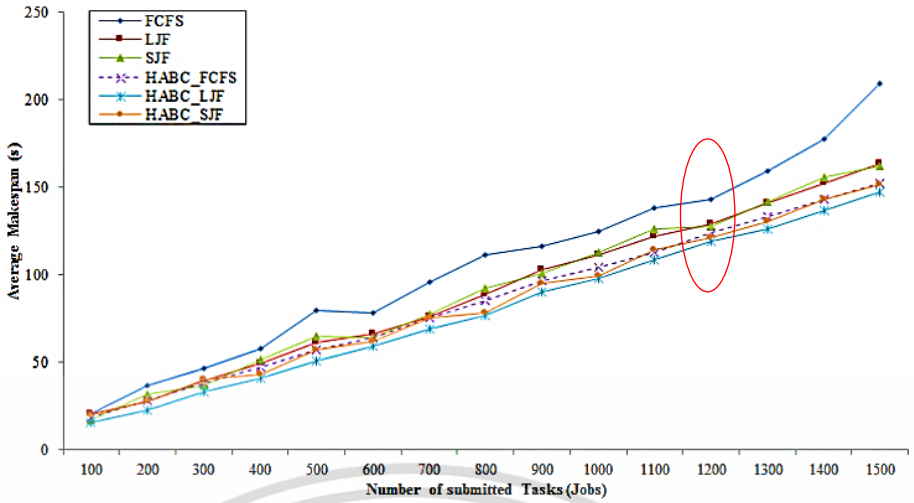
สุด (min) เวลาที่ใช้ในการทำงานรวมมากสุด (max) และเวลาที่ใช้ในการทำงานรวมเฉลี่ย (avg) ซึ่งให้ค่าของเวลาที่ต่ำที่สุด เมื่อเปรียบเทียบกับการใช้พารามิเตอร์ใน Case I และ Case II

กรณีที่ 2: เปรียบเทียบเวลารวมเฉลี่ยในการทำงานโดยใช้ชุดข้อมูลประเภทแบบสุ่มอิสระที่มีการกระจายข้อมูลแบบแจกแจงปกติ

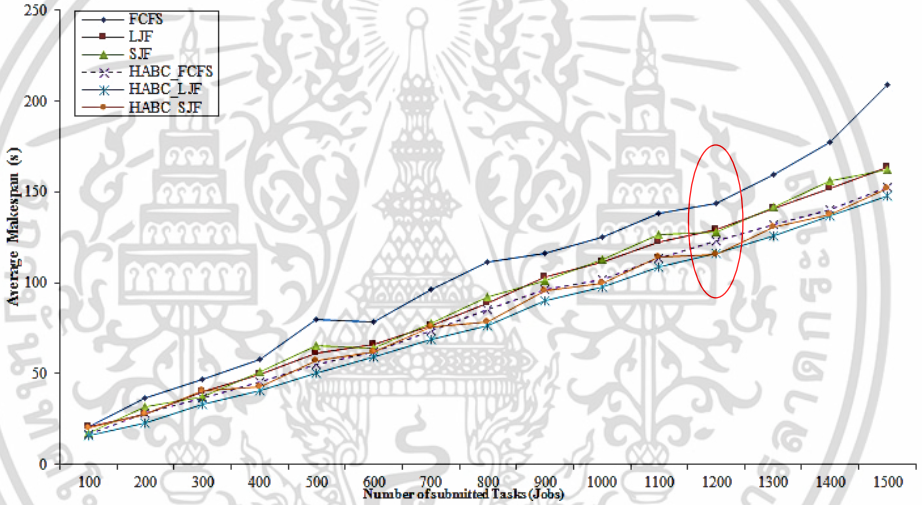
การเปรียบเทียบเวลารวมเฉลี่ยในการทำงานกับจำนวนงานที่มีการเปลี่ยนแปลงในระบบกับชุดข้อมูลสุ่มแบบอิสระ โดยใช้ชุดข้อมูล D_4 เพื่อประเมินประสิทธิภาพของวิธีการที่นำเสนอ (HABC) ซึ่งผลจากการทดสอบความสามารถในการทำงานโดยเปรียบเทียบกับจำนวนงานที่มีการเปลี่ยนแปลงและจำนวนผึ้งในระบบมีการเปลี่ยนแปลง แสดงได้ดังตารางที่ 4.7 และรูปที่ 4.4 – 4.6

ตารางที่ 4.7 เวลาในการทำงานรวมโดยใช้ชุดข้อมูลประเภทแบบการกระจายข้อมูลแบบแจกแจงปกติ

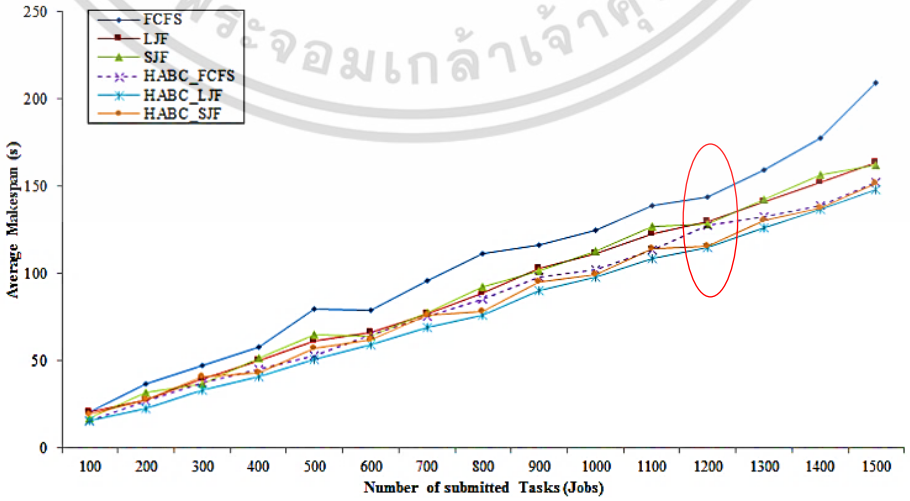
Task	Algorithm	Processing Time (s)								
		Case I			Case II			Case III		
		min	*max*	*avg*	*min*	*max*	*avg*	*min*	*max*	*avg*
100	HABC_FCFS	16.51	20.43	18.24	**13.74**	19.65	16.49	**13.93**	15.89	**15.21**
	HABC_LJF	**14.97**	**16.69**	**15.82**	15.44	**15.89**	**15.62**	14.68	**15.66**	15.50
	HABC_SJF	18.92	20.54	19.74	16.11	20.49	19.66	16.10	20.49	19.02
200	HABC_FCFS	24.02	30.55	28.16	24.90	31.68	28.00	22.87	27.53	26.49
	HABC_LJF	**22.20**	**22.89**	**22.55**	**22.21**	**22.44**	**22.34**	**22.22**	**22.41**	**22.34**
	HABC_SJF	26.76	29.12	27.63	26.95	27.84	27.68	27.41	27.84	27.73
300	HABC_FCFS	**32.58**	44.01	38.29	**31.06**	42.82	36.66	**32.80**	38.81	37.23
	HABC_LJF	33.20	**33.62**	**33.40**	33.20	**33.43**	**33.34**	33.25	**33.46**	**33.35**
	HABC_SJF	39.27	41.06	40.45	39.93	40.85	40.65	40.09	40.83	40.57
400	HABC_FCFS	43.96	52.57	47.13	42.56	51.90	45.58	42.31	45.49	44.78
	HABC_LJF	**40.51**	**41.21**	**40.77**	**40.51**	**40.68**	**40.58**	**40.50**	**40.64**	**40.57**
	HABC_SJF	41.90	43.25	42.88	42.38	42.99	42.81	42.57	43.01	42.84
500	HABC_FCFS	52.66	59.81	56.96	50.99	58.90	55.06	51.19	54.13	52.93
	HABC_LJF	**50.32**	**50.67**	**50.49**	**50.28**	**50.48**	**50.41**	**50.20**	**50.52**	**50.38**
	HABC_SJF	56.75	57.71	57.29	56.78	57.29	57.11	56.60	57.30	57.13
600	HABC_FCFS	60.07	68.58	64.29	59.43	65.75	62.10	**57.61**	67.18	64.65
	HABC_LJF	**59.04**	**59.42**	**59.20**	**59.05**	**59.19**	**59.13**	58.93	**59.22**	**59.08**
	HABC_SJF	61.55	62.15	61.81	61.70	62.10	61.90	61.69	62.10	61.87
700	HABC_FCFS	71.02	80.93	75.70	**68.52**	77.42	73.49	72.29	76.13	75.00
	HABC_LJF	**68.90**	**69.28**	**69.08**	68.87	**69.00**	**68.94**	**68.88**	**69.03**	**68.95**
	HABC_SJF	75.13	76.26	75.84	75.02	76.12	75.76	75.19	76.14	75.76
800	HABC_FCFS	81.39	89.54	85.64	82.16	89.04	84.97	82.08	87.41	85.23
	HABC_LJF	**76.12**	80.69	**76.94**	**76.12**	**76.42**	**76.24**	**76.04**	**76.36**	**76.26**
	HABC_SJF	77.82	**78.88**	78.46	78.01	78.50	78.27	77.96	78.54	78.28
900	HABC_FCFS	91.29	100.69	96.72	93.86	99.71	95.93	92.45	101.71	97.68
	HABC_LJF	**89.80**	**90.32**	**90.10**	**89.84**	**90.14**	**89.99**	**89.85**	**90.05**	**89.97**
	HABC_SJF	94.48	95.54	95.10	94.89	95.47	95.33	94.89	95.44	95.25
1000	HABC_FCFS	97.99	109.36	104.12	97.94	107.19	102.07	98.09	104.03	102.22
	HABC_LJF	**97.85**	**98.15**	**97.99**	**97.74**	**97.99**	**97.87**	**97.80**	**97.94**	**97.86**
	HABC_SJF	98.95	99.75	99.36	99.10	99.79	99.56	99.18	99.77	99.56
1100	HABC_FCFS	109.30	116.83	113.02	**108.12**	117.74	113.36	109.81	114.99	113.58
	HABC_LJF	**108.66**	**109.00**	**108.86**	108.59	**108.87**	**108.72**	**108.65**	**108.87**	**108.77**
	HABC_SJF	113.59	114.59	114.13	113.79	114.53	114.34	114.01	114.52	114.28
1200	HABC_FCFS	121.24	128.15	124.03	118.80	128.18	123.14	125.27	129.50	127.73
	HABC_LJF	**115.09**	**119.90**	119.44	**114.79**	119.57	116.26	**114.83**	**115.04**	**114.93**
	HABC_SJF	115.52	126.92	**117.57**	115.29	**116.04**	**115.85**	115.51	116.09	115.83
1300	HABC_FCFS	128.64	137.11	133.07	128.51	134.98	132.14	128.02	133.16	132.27
	HABC_LJF	**125.81**	**126.13**	**125.97**	**125.77**	**125.96**	**125.86**	**125.79**	**125.97**	**125.87**
	HABC_SJF	129.99	131.03	130.52	130.31	130.78	130.58	130.07	130.76	130.57
1400	HABC_FCFS	138.21	149.02	142.97	**136.44**	144.74	140.61	136.50	140.55	138.73
	HABC_LJF	**136.75**	**137.06**	**136.93**	136.73	**137.00**	**136.87**	**136.80**	**136.99**	**136.89**
	HABC_SJF	137.31	144.59	143.49	136.97	137.72	137.47	136.81	137.78	137.43
1500	HABC_FCFS	149.75	157.66	152.49	150.41	156.97	152.81	150.25	150.25	151.93
	HABC_LJF	**147.45**	**147.86**	**147.63**	**147.45**	**147.66**	**147.55**	**147.46**	**147.46**	**147.56**
	HABC_SJF	151.38	151.96	151.70	151.53	152.04	151.80	151.54	151.54	151.77



รูปที่ 4.4 เวลาทำงานรวมเฉลี่ยของชุดข้อมูล D_4 เมื่อกำหนดค่าพารามิเตอร์ของ ABC โดยใช้ Case I



รูปที่ 4.5 เวลาทำงานรวมเฉลี่ยของชุดข้อมูล D_4 เมื่อกำหนดค่าพารามิเตอร์ของ ABC โดยใช้ Case II



รูปที่ 4.6 เวลาทำงานรวมเฉลี่ยของชุดข้อมูล D_4 เมื่อกำหนดค่าพารามิเตอร์ของ ABC โดยใช้ Case III

จากรูปที่ 4.4 - 4.6 และตารางที่ 4.7 แสดงให้เห็นถึงเวลาที่ใช้ในการประมวลผลทั้งหมดในกรณีที่เครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมที่แตกต่างกัน ในกรณีของชุดข้อมูลสุ่มที่มีการกระจายของงานแบบการแจกแจงปกติ ผลจากการทดลองแสดงให้เห็นว่า HABC_LJF ให้ผลลัพธ์ที่ดีที่สุดในการจัดตารางงานเข้าทำงาน เมื่อปริมาณงานในระบบมีปริมาณเพิ่มมากขึ้น นอกจากนี้เมื่อทำการเปรียบเทียบในแง่ของเวลารวมน้อยสุดในการงาน เวลารวมที่มากที่สุดในการทำงาน และค่าเวลาเฉลี่ยที่ใช้ในการทำงานยังพบว่า HABC_LJF ยังมีคงมีประสิทธิภาพในการทำงานดีที่สุด และเมื่อพิจารณาในกรณีของการหาค่าพารามิเตอร์ที่เหมาะสมที่สุดของ HABC ดังที่แสดงในตารางที่ 4.4 พบว่า Case III ให้ค่าพารามิเตอร์ที่เหมาะสมที่สุด

ในทางกลับกัน การตั้งค่าพารามิเตอร์ใน Case I และ Case II ส่งผลต่อการทดลองนั้น เช่น เมื่อกำหนดจำนวนงาน 1200 งาน พบว่า HABC_SJF นั้นให้เวลารวมในการประมวลผลที่น้อยที่สุด ซึ่งขัดแย้งกับผลลัพธ์ก่อนหน้าและให้ผลลัพธ์ที่แตกต่างจาก Case III ซึ่งให้ผลที่คล้ายคลึงกับการทดลองกรณีที่ 1 และเมื่อทำการทดลองโดยการเพิ่มจำนวนผึ้ง พบว่าไม่มีความแตกต่างอย่างมีนัยสำคัญในเวลาที่ใช้ในการประมวลผลซึ่งให้ผลที่ใกล้เคียงกับการกำหนดพารามิเตอร์โดยใช้ Case III

4.5 ผลการทดลองเพื่อประเมินประสิทธิภาพของการจัดตารางงานโดยใช้ขั้นตอนอาณานิคมผึ้งเทียมควบคู่กับการจัดตารางงานแบบฮิวริสติก

การประเมินประสิทธิภาพในการจัดตารางงานในเครื่องคอมพิวเตอร์เสมือน โดยใช้ขั้นตอนอาณานิคมผึ้งเทียมควบคู่กับการจัดลำดับงานแบบฮิวริสติก (HABC) โดยทำการเปรียบเทียบกับวิธีการจัดตารางงานโดยใช้วิธีทางเมตาฮิวริสติก (Meta-heuristic algorithm) โดยได้ทำการเลือกวิธีการที่ได้รับความนิยม เช่น การหาค่าความเหมาะสมที่สุดด้วยวิธีอาณานิคมมด (ACO algorithm) และขั้นตอนวิธีการหาคำตอบเหมาะสมที่สุดแบบกลุ่มอนุภาค (PSO algorithm)

4.5.1 การตั้งค่าพารามิเตอร์สภาพแวดล้อมในการทดลอง

4.5.1.1 กรณีสภาพแวดล้อมภายในเครื่องคอมพิวเตอร์เสมือนเหมือนกัน

การตั้งค่าพารามิเตอร์แบบจำลองเพื่อประเมินประสิทธิภาพของระบบการประมวลผลแบบกลุ่มเมฆ กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในเหมือนกันทั้งหมด แสดงค่าดังตารางที่ 4.8

ตารางที่ 4.8 ค่าพารามิเตอร์ของแบบจำลองการประมวลผลแบบกลุ่มเมฆ กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมเหมือนกัน

Type	Parameter	Value
Datacenter	Number of Datacenter	10
	Number of Host	5

ตารางที่ 4.8 ค่าพารามิเตอร์ของแบบจำลองการประมวลผลแบบกลุ่มเมฆ กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมเหมือนกัน (ต่อ)

Type	Parameter	Value
Datacenter	Type of Manager	Space_Shared, Time_Shared
	Number of PE per Host	4
	MIPS of PE	2000
	Host Memory (RAM) (MB)	10240
	Datacenter Cost	10
Virtual Machine (VMs)	Total number of VM	50
	MIPS of PE	9726
	Number of PE per VM	1
	VM Memory (RAM) (MB)	1024
	Bandwidth (Bit)	5000
	Type of Manager	Time_Shared
Task	Total number of Task	100 – 1500
	Length of Task (MI)	5000 - 45000
	Number of PE per requirement	1
	Type of Manager	Space_Shared

4.5.1.2 กรณีสภาพแวดล้อมภายในเครื่องคอมพิวเตอร์เสมือนแตกต่างกัน

การตั้งค่าพารามิเตอร์แบบจำลองเพื่อประเมินประสิทธิภาพของระบบการประมวลผลแบบกลุ่มเมฆ กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในแตกต่างกัน แสดงได้ดังตารางที่ 4.9

ตารางที่ 4.9 ค่าพารามิเตอร์ของแบบจำลองการประมวลผลแบบกลุ่มเมฆ กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมแตกต่างกัน

Type	Parameter	Value
Datacenter	Number of Datacenter	10
	Number of Host	5
	Type of Manager	Space_Shared, Time_Shared
	Number of PE per Host	4
	MIPS of PE	2000

ตารางที่ 4.9 ค่าพารามิเตอร์ของแบบจำลองการประมวลผลแบบกลุ่มเมฆ กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมแตกต่างกัน (ต่อ)

Type	Parameter	Value
Datacenter	Number of Datacenter	10
	Number of Host	5
	Type of Manager	Space_Shared, Time_Shared
	Number of PE per Host	4
	MIPS of PE	2000
	Host Memory (RAM) (MB)	10240
	Datacenter Cost	10
Virtual Machine (VMs)	Total number of VM	50
	MIPS of PE	5000 - 9726
	Number of PE per VM	2
	VM Memory (RAM) (MB)	512 - 4096
	Bandwidth (Bit)	1000 - 10000
	Type of Manager	Time_Shared
Task	Total number of Task	100 – 1500
	Length of Task (MI)	5000 - 45000
	Number of PE per requirement	1
	Type of Manager	Space_Shared

4.5.2 การตั้งค่าพารามิเตอร์ใน HABC, ACO, PSO และ IPSO

ในการทดลองครั้งนี้ ได้กำหนดค่าพารามิเตอร์ของประชากรและเงื่อนไขต่างๆ โดยอ้างอิงพารามิเตอร์ตามบทความที่เกี่ยวข้อง ในการประเมินประสิทธิภาพของวิธีการที่นำเสนอโดยทำการเปรียบเทียบกับวิธีการ HABC [74], ACO [56,75], PSO [76] และ IPSO [77] ทั้งนี้เป็นที่ทราบกันดีว่าพารามิเตอร์เหล่านี้มีผลอย่างมากต่อประสิทธิภาพการทำงานของอัลกอริทึม ทั้งนี้การกำหนดพารามิเตอร์จะแตกต่างกันไปตามขนาดและลักษณะของปัญหา ในการทดลองได้กำหนดพารามิเตอร์ของผึ้งที่เหมาะสมที่สุดสำหรับอัลกอริทึมที่นำเสนอ ดังที่ได้เสนอผลการทดลองในหัวข้อที่ 4.4 พารามิเตอร์ที่ใช้ในการตั้งค่าแสดงได้ดังตารางที่ 4.10

ตารางที่ 4.10 ค่าของพารามิเตอร์ที่ใช้ในวิธีการที่นำเสนอและอัลกอริทึมอื่นที่นำมาเปรียบเทียบ

HABC		ACO		PSO		IPSO	
Parameter	**Values**	**Parameter**	**Values**	**Parameter**	**Values**	**Parameter**	**Values**
Number of scout bees (n)	960	Ants	50	Number of particles	100	Population size	100
Number of sites selected out of n visited sites (m)	96	Maximum probability of trial	0.98	Weight (w)	0.9	Weight (w_{min}, w_{max})	0.1, 0.9
Number of best sites out of m selected sites (e)	1	Local search probability	0.01	Acceleration factor (C_1)	1.49445	Acceleration factor (C_1)	1.49445
Number of bees recruited for searching best e sites (nep)	768	Evaporation rate	0.01	Acceleration factor (C_2)	1.49445	Acceleration factor (C_2)	1.49445
Number of bees recruited for searching other (m-e) selected sites (nsp)	192	Max number of iterations	1000	Max number of iterations	1000	Maximum iterations	1000
Max number of iterations	1000					K	5

HABC, Heuristic Task Scheduling with Artificial Bee Colony; ACO, Ant Colony Optimization; PSO, Particle Swarm Optimization; IPSO, improved PSO.

4.5.3 ผลการทดลองเพื่อประเมินประสิทธิภาพในการทำงาน

จากการทดลองในหัวข้อ 4.4 ได้แสดงให้เห็นถึงประสิทธิภาพของอัลกอริทึมที่นำเสนอควบคู่กับการจัดตารางงานโดยใช้วิธีการทางฮิวริสติก 3 วิธี คือ HABC_LJF, HABC_FCFS และ HABC_SJF ถูกนำมาเปรียบเทียบกับการจัดตารางโดยใช้ฮิวริสติกเพียงอย่างเดียว คือ LJF, FCFS และ SJF ผลจากการทดลองก่อนหน้าพบว่า HABC_LJF ให้ประสิทธิภาพในการจัดตารางงานที่ดีกว่าวิธีการอื่นที่นำมาเปรียบเทียบ อีกทั้งยังสามารถกระจายงานแบบสมดุลไปยังเครื่องคอมพิวเตอร์เสมือนและให้เวลาในการประมวลผลของงานน้อยที่สุด ในการทดลองครั้งนี้ได้ทำการทดลองกับสภาพแวดล้อมภายในของเครื่องคอมพิวเตอร์ 2 แบบคือ แบบที่สภาพแวดล้อมภายในของเครื่องคอมพิวเตอร์เสมือนเหมือนกันทั้งหมด (Homogeneous) และแตกต่างกันทั้งหมด (Heterogeneous) เพื่อประเมินประสิทธิภาพของ HABC_LJF โดยใช้ชุดข้อมูลที่มีรูปแบบแตกต่างกัน อีกทั้งยังทำการเปรียบเทียบกับอัลกอริทึมการจัดตารางงานวิธีการอื่นๆ

ในการทดลองกำหนดให้จำนวนเครื่องคอมพิวเตอร์เสมือนมีค่าคงที่เท่ากับ 50 เครื่อง และทดสอบความสามารถในการทำงานเปรียบเทียบกับจำนวนงานที่มีการเปลี่ยนแปลง โดยกำหนดให้จำนวนงานเพิ่มขึ้นครั้งละ 100 งาน เริ่มจาก 100 – 1500 งาน

การประเมินประสิทธิภาพของการจัดตารางงานโดยใช้วิธี HABC_LJF ในระบบการประมวลผลแบบกลุ่มเมฆได้แบ่งการทดลองเพื่อประเมินประสิทธิภาพการทำงานออกเป็น 3 ส่วนดังนี้ คือ ส่วนที่ 1 เวลาการทำงานรวมเฉลี่ย (Makespan) ส่วนที่ 2 ค่าความไม่สมดุลของการกระจายข้อมูล (Degree of Imbalance: DI) และส่วนที่ 3 ค่าเบี่ยงแบนมาตรฐาน (Standard Deviation: S.D.) มีรายละเอียดดังต่อไปนี้

4.5.3.1 ผลการทดสอบประสิทธิภาพในส่วนของเวลาการทำงานรวมเฉลี่ย

การประเมินประสิทธิภาพของ HABC_LJF algorithm ในระบบการประมวลผลแบบกลุ่มเมฆ ได้แบ่งการทดลองออกเป็น 4 ส่วนตามประเภทของชุดข้อมูลที่ใช้ในการทดสอบ ดังแสดงในตารางที่ 4.1 ผลการทดลองของ 4 ประเภทตามชุดข้อมูลสามารถแสดงรายละเอียดได้ดังนี้

กรณีที่ 1: ชุดข้อมูลที่มีการกระจายแบบสุ่มอิสระ (Random)

การประเมินประสิทธิภาพในการทำงานเปรียบเทียบกับจำนวนงานที่มีการเปลี่ยนแปลง เมื่อจำนวนเครื่องคอมพิวเตอร์เสมือนมีค่าคงที่ ในการทดลองใช้ชุดข้อมูลที่แสดงในตารางที่ 4.1 โดยใช้ชุดข้อมูล $D_1 - D_3$ คือชุดข้อมูลแบบสุ่มอิสระที่ไม่มีข้อกำหนดของรูปแบบที่แน่นอน ซึ่งแต่ละชุดข้อมูลนั้นมีการกำหนดช่วงขนาดของงานที่แตกต่างกัน เพื่อทำการเปรียบเทียบความสามารถในการจัดตารางงานของโดยใช้ HABC_LJF เปรียบเทียบกับวิธีการจัดตารางงานโดยใช้ ACO, PSO และ IPSO ในกรณีที่สภาพแวดล้อมภายในเครื่องเสมือนเหมือนกันและแตกต่างกัน ผลการทดลองแสดงได้ดังตารางที่ 4.11 และตารางที่ 4.12

ตารางที่ 4.11 เวลาการทำงานรวมของชุดข้อมูลที่ $D_1 - D_3$ เมื่อเปรียบเทียบระหว่าง HABC, ACO, PSO และ IPSO (Homogenous)

Dataset	Task	Variable	HABC_LJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_FCFS	IPSO_LJF	IPSO_SJF
D_1	300	Min	**8.623**	9.600	9.023	9.982	25.982	25.030	25.707	29.578	27.448	27.979
		Max	**8.643**	10.592	9.343	11.017	29.017	26.329	26.408	32.381	31.162	34.342
		Avg	**8.633**	10.227	9.225	10.324	26.988	25.563	25.876	30.626	28.766	30.273
	900	Min	**25.015**	26.352	25.739	26.702	83.733	81.206	83.467	107.237	103.885	105.384
		Max	**25.262**	27.369	26.042	27.561	84.777	96.767	86.560	113.512	108.464	107.781
		Avg	**25.073**	26.819	25.881	26.976	84.165	89.705	84.255	109.620	105.725	106.302
	1500	Min	**42.275**	43.641	42.892	44.050	151.965	153.637	153.482	215.075	211.427	303.934
		Max	**42.614**	45.020	43.523	45.058	164.660	156.171	157.010	229.624	315.327	319.373
		Avg	**42.409**	44.233	43.228	44.641	158.947	154.736	154.529	222.093	262.953	310.891
D_2	300	Min	**16.308**	18.685	18.036	19.148	36.465	32.567	33.462	37.476	35.041	37.075
		Max	**16.341**	19.719	18.288	20.252	39.547	34.134	35.205	39.296	42.372	39.112
		Avg	**16.319**	19.258	18.161	19.646	38.194	32.993	33.906	38.645	38.841	37.428
	900	Min	**48.340**	50.969	50.150	51.526	108.373	105.943	106.805	129.830	128.848	127.957
		Max	**48.907**	52.823	51.427	53.980	109.092	108.074	108.557	137.917	132.341	130.492
		Avg	**48.475**	51.795	50.353	52.154	108.704	106.622	107.725	133.191	130.587	129.206
	1500	Min	**80.237**	82.888	81.959	83.918	192.893	189.624	194.519	250.603	245.937	248.938
		Max	**81.623**	84.951	83.743	86.442	193.491	192.754	198.249	263.329	256.496	254.803
		Avg	**80.516**	84.122	82.818	85.263	193.206	191.226	196.048	255.369	251.500	251.033
D_3	300	Min	**22.229**	25.645	24.983	26.244	40.877	38.553	39.606	45.246	45.293	46.121
		Max	**22.262**	27.500	25.278	26.867	42.182	41.206	41.803	56.463	50.604	50.163
		Avg	**22.244**	26.378	25.126	26.529	41.230	39.237	40.155	49.921	47.083	47.706
	900	Min	**67.077**	70.642	70.006	71.203	127.109	125.271	125.686	158.775	158.025	160.241
		Max	**67.245**	73.066	72.770	74.383	128.451	128.572	128.420	176.779	174.420	169.610
		Avg	**67.188**	71.676	70.506	72.115	127.780	126.110	126.468	169.075	165.177	164.849
	1500	Min	**111.770**	116.333	114.901	116.010	223.859	222.317	223.264	290.299	290.595	290.642
		Max	**112.050**	119.118	118.887	119.478	227.468	225.851	225.024	304.952	289.595	300.847
		Avg	**111.870**	117.533	116.918	118.256	225.457	223.576	224.219	296.090	293.595	294.004

จากผลการทดลองในตารางที่ 4.11 แสดงเวลาทำงานรวมเฉลี่ย (Avg makespan) เวลาทำงานรวมที่น้อยที่สุด (Min makespan) และเวลาทำงานรวมที่มากที่สุด (Max makespan) โดยเปรียบเทียบการจัดตารางกรณีสภาพแวดล้อมภายในเครื่องเสมือนเหมือนกัน โดยใช้ HABC_LJF, ACO, PSO, และ IPSO ผลการทดสอบประสิทธิภาพแสดงให้เห็นว่า HABC_LJF มีประสิทธิภาพในการจัดตารางงานที่ดีที่สุด เมื่อปริมาณงานในระบบมีปริมาณเพิ่มมากขึ้น

นอกจากนี้ เมื่อมองในมุมของการกำหนดช่วงขนาดของงานที่แตกต่างกัน โดยทำการทดสอบกับชุดข้อมูลที่มีช่วงขนาดของงานที่แตกต่างกัน คือ ชุดข้อมูล $D_1 - D_3$ ผลจากการทดลองแสดงให้เห็นว่า HABC_LJF มีประสิทธิภาพในการจัดตารางงานที่เหนือกว่าอัลกอริทึมอื่นที่นำมาเปรียบเทียบ แม้ช่วงขนาดของงานมีการเปลี่ยนแปลง ในการทดสอบกับสภาพแวดล้อมภายในเครื่องเสมือนที่เหมือนกัน

ตารางที่ 4.12 เวลาการทำงานรวมของชุดข้อมูลที่ D_1 – D_3 เมื่อเปรียบเทียบระหว่าง HABC, ACO, PSO และ IPSO (Heterogenous)

Dataset	Task	Variable	HABC_LJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_FCFS	IPSO_LJF	IPSO_SJF
D_1	300	Min	**7.320**	7.982	7.500	8.292	24.207	23.707	25.749	27.498	26.623	27.055
		Max	**7.368**	9.803	8.181	9.925	27.171	24.370	26.155	30.966	29.537	27.616
		Avg	**7.345**	8.817	7.765	9.086	25.610	23.996	26.007	28.792	27.144	27.322
	900	Min	**20.601**	21.133	20.605	21.707	80.445	79.467	81.114	101.135	98.005	99.690
		Max	**20.865**	23.311	21.808	22.592	82.856	81.043	88.297	106.207	111.467	101.623
		Avg	**20.715**	22.256	20.941	22.146	81.698	79.943	82.409	102.858	100.261	100.695
	1500	Min	**34.213**	34.976	34.218	35.310	148.688	148.288	148.757	216.645	213.903	219.464
		Max	**34.717**	37.461	35.203	35.970	152.961	150.648	151.124	247.682	256.862	247.018
		Avg	**34.450**	36.114	34.700	35.630	150.867	149.111	149.999	227.404	232.807	231.975
D_2	300	Min	**14.595**	15.385	14.935	15.933	32.330	30.390	33.420	32.324	32.507	34.153
		Max	**14.687**	18.406	15.641	17.333	36.045	30.760	33.823	34.935	34.675	35.080
		Avg	**14.615**	16.978	15.316	16.839	33.353	30.545	33.666	33.442	33.155	34.533
	900	Min	**40.299**	40.305	40.486	41.219	98.444	96.444	98.889	118.667	118.246	119.673
		Max	**40.706**	44.432	41.102	43.978	101.081	97.356	100.745	131.408	123.997	124.640
		Avg	**40.439**	42.816	40.821	42.402	99.779	96.823	99.582	124.194	119.795	121.197
	1500	Min	**65.784**	66.160	66.063	66.240	175.483	175.397	169.006	229.331	230.574	233.232
		Max	**67.162**	69.771	67.677	68.459	181.691	178.763	204.867	261.110	261.855	237.858
		Avg	**66.243**	67.947	66.418	67.835	177.890	177.211	180.840	239.835	237.419	234.780
D_3	300	Min	**20.374**	21.310	20.541	22.037	36.164	35.042	38.526	38.310	36.574	40.152
		Max	**20.595**	25.453	21.479	23.777	41.005	35.376	39.248	42.268	39.352	40.698
		Avg	**20.412**	23.130	21.084	22.705	38.605	35.183	38.678	39.821	38.069	40.422
	900	Min	**56.053**	56.317	56.796	57.418	114.889	108.005	109.332	128.071	125.720	129.099
		Aax	**56.986**	61.682	57.280	59.408	118.975	109.462	112.828	135.651	129.380	131.149
		Avg	**56.632**	58.188	57.014	58.687	116.954	108.515	110.325	130.650	127.624	130.120
	1500	Min	**92.419**	93.852	93.374	93.751	195.333	196.762	191.022	264.640	244.018	245.887
		Max	**92.945**	96.593	93.358	94.951	222.115	219.735	195.878	316.860	278.675	251.534
		Avg	**92.624**	95.327	93.005	94.492	205.862	207.061	192.619	285.370	256.390	248.354

ผลการทดลองกรณีสภาพแวดล้อมภายในเครื่องเสมือนแตกต่างกัน แสดงได้ดังตารางที่ 4.12 ผลจากการทดลองพบว่า HABC_LJF มีประสิทธิภาพการทำงานที่ดีที่สุด ในแง่ของเวลาทำงานรวมเฉลี่ย เวลาทำงานรวมที่น้อยที่สุด และเวลาทำงานรวมที่มากที่สุด นอกจากนี้เมื่อใช้ชุดข้อมูลที่มีช่วงขนาดของงานที่แตกต่างกัน พบว่า HABC_LJF ยังคงมีประสิทธิภาพเหนือกว่าอัลกอริทึมอื่นที่นำมาเปรียบเทียบ เช่นเดียวกับการทดสอบในกรณีที่สภาพแวดล้อมภายในเครื่องเสมือนที่เหมือนกัน

กรณีที่ 2: ชุดข้อมูลที่มีการกระจายแบบการแจกแจงปกติ

การประเมินประสิทธิภาพในการทำงานเปรียบเทียบกับจำนวนงานที่มีการเปลี่ยนแปลง เมื่อจำนวนเครื่องคอมพิวเตอร์เสมือนมีค่าคงที่ ในการทดลองใช้ชุดข้อมูลที่แสดงในตารางที่ 4.1 โดยใช้ชุดข้อมูล D_4 – D_6 คือชุดข้อมูลแบบสุ่มที่มีการกระจายข้อมูลแบบแจกแจงปกติ ซึ่งแต่ละชุดข้อมูลนั้นมีการกำหนดช่วงขนาดของงานที่แตกต่างกัน เพื่อทำการเปรียบเทียบความสามารถในการจัดตารางงานโดยใช้วิธี HABC_LJF เปรียบเทียบกับการจัดตารางงานโดยใช้วิธี ACO, PSO, และ IPSO ในกรณีที่สภาพแวดล้อมภายในเครื่องเสมือนเหมือนกันและแตกต่างกันทั้งหมด ผลการทดลองแสดงได้ดังตารางที่ 4.13 และตารางที่ 4.14

จากผลการทดลองในตารางที่ 4.13 แสดงเวลาทำงานรวมเฉลี่ย เวลาทำงานรวมที่น้อยที่สุด และเวลาทำงานรวมที่มากที่สุด โดยเปรียบเทียบการจัดตารางกรณีสภาพแวดล้อมภายในเครื่องเสมือนเหมือนกัน โดยใช้วิธี HABC_LJF, ACO, PSO, และ IPSO ผลการทดสอบประสิทธิภาพแสดงให้เห็นว่า HABC_LJF มีประสิทธิภาพในการจัดตารางงานที่ดีที่สุด เมื่อปริมาณงานในระบบมี

ปริมาณเพิ่มมากขึ้น แม้ว่าขนาดของงานมีการเปลี่ยนแปลง การจัดตารางงานโดยใช้ HABC_LJF ยังคงสามารถทำงานได้อย่างมีประสิทธิภาพเช่นเดิม

ตารางที่ 4.13 เวลาการทำงานรวมของชุดข้อมูลที่ D_4 – D_6 เมื่อเปรียบเทียบระหว่าง HABC, ACO, PSO และ IPSO (Homogenous)

Dataset	Task	Variable	HABC_LJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_FCFS	IPSO_LJF	IPSO_SJF
D_4	300	Min	**8.575**	9.883	9.111	10.170	25.540	24.501	25.472	29.994	28.758	28.635
		Max	**8.634**	11.052	9.501	10.740	26.333	24.940	26.308	32.600	31.167	29.165
		Avg	**8.584**	10.331	9.308	10.434	25.798	24.643	25.813	30.624	29.298	28.851
	900	Min	**25.582**	27.020	26.319	27.101	82.715	81.997	84.421	110.198	107.394	107.813
		Max	**25.867**	27.866	26.746	28.302	92.448	84.768	87.776	119.204	110.113	110.444
		Avg	**25.669**	27.400	26.511	27.644	84.544	83.432	85.733	113.127	108.915	109.174
	1500	Min	**41.993**	43.462	42.731	43.983	149.972	152.230	151.124	221.121	215.575	218.054
		Max	**42.246**	44.370	44.234	44.703	157.576	154.169	161.643	233.171	220.500	221.989
		Avg	**42.140**	43.799	43.177	44.364	152.614	153.111	153.875	225.417	217.941	219.824
D_5	300	Min	**16.182**	18.335	17.793	18.982	34.274	32.607	33.625	37.653	38.349	37.446
		Max	**16.215**	20.528	19.076	20.624	35.227	34.275	34.922	42.583	40.628	37.899
		Avg	**16.194**	19.098	18.004	19.429	34.484	32.893	33.952	40.022	38.938	37.747
	900	Min	**48.367**	51.071	50.232	51.457	108.799	106.392	107.270	135.063	129.111	130.951
		Max	**48.499**	52.978	51.790	54.115	118.703	108.260	108.692	138.506	134.602	134.658
		Avg	**48.404**	52.034	50.572	52.473	110.707	107.140	107.825	136.418	132.357	132.098
	1500	Min	**80.594**	83.666	82.612	83.878	193.607	192.516	192.841	258.106	252.917	254.743
		Max	**80.808**	85.558	84.510	86.401	256.281	194.782	194.970	285.920	270.308	258.800
		Avg	**80.701**	84.506	83.578	85.431	207.942	193.393	193.883	263.625	258.098	256.448
D_6	300	Min	**22.266**	25.403	24.843	26.012	40.675	38.361	39.239	45.958	45.530	44.446
		Max	**22.322**	27.499	25.273	27.442	40.989	43.362	46.370	47.159	49.020	45.375
		Avg	**22.275**	26.316	25.094	26.585	40.832	39.170	41.054	46.464	46.335	44.698
	900	Min	**66.586**	70.473	69.570	70.620	125.921	123.478	123.851	151.630	150.560	149.820
		Max	**66.733**	72.472	71.977	73.627	126.799	126.650	127.080	157.917	157.273	151.973
		Avg	**66.675**	71.373	70.054	71.545	126.492	124.704	125.075	154.434	152.390	151.096
	1500	Min	**111.167**	115.941	114.366	115.366	223.019	221.505	222.173	287.264	281.794	285.184
		Max	**111.508**	117.362	117.450	118.750	226.545	259.360	224.472	297.683	292.236	293.692
		Avg	**111.291**	116.548	116.070	117.391	224.414	238.211	223.379	291.949	288.023	287.741

ตารางที่ 4.14 เวลาการทำงานรวมของชุดข้อมูลที่ D_4 – D_6 เมื่อเปรียบเทียบระหว่าง HABC, ACO, PSO และ IPSO (Heterogenous)

Dataset	Task	Variable	HABC_LJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_FCFS	IPSO_LJF	IPSO_SJF
D_4	300	Min	**7.347**	8.266	7.460	8.378	25.131	23.905	24.550	28.075	28.058	28.192
		Max	**7.536**	9.874	7.993	9.678	27.318	25.652	25.811	33.277	29.661	30.436
		Avg	**7.429**	9.157	7.711	9.006	25.850	24.791	24.849	30.191	28.811	28.953
	900	Min	**20.822**	21.427	20.973	22.206	79.248	81.988	77.327	98.817	96.358	98.507
		Max	21.869	23.342	**21.397**	23.313	81.374	85.940	78.608	103.967	98.902	101.255
		Avg	**21.215**	22.360	21.226	22.597	80.299	84.128	77.870	100.968	97.827	99.515
	1500	Min	**33.862**	34.144	33.987	34.943	141.381	140.255	140.594	199.534	195.573	197.259
		Max	**34.482**	36.491	35.618	35.978	162.631	153.213	144.570	211.162	201.047	202.690
		Avg	**34.210**	35.305	34.399	35.434	151.768	144.389	142.179	203.859	198.525	199.162
D_5	300	Min	**14.211**	14.876	14.837	16.105	25.131	23.905	24.550	33.591	32.701	34.567
		Max	**14.306**	18.346	15.362	17.862	27.318	25.652	25.811	37.458	33.676	35.061
		Avg	**14.261**	16.768	15.100	16.640	25.850	24.791	24.849	35.377	33.052	34.751
	900	Min	**40.087**	41.393	40.543	41.148	79.248	81.988	77.327	116.100	114.003	117.011
		Max	**40.735**	47.170	41.090	42.831	81.374	85.940	78.608	122.000	116.834	118.787
		Avg	**40.256**	43.287	40.754	42.407	80.299	84.128	77.870	118.567	115.730	117.796
	1500	Min	**66.087**	66.979	66.228	66.660	141.381	140.255	140.594	231.830	228.575	230.755
		Max	**66.948**	70.342	67.070	72.701	162.631	153.213	144.570	242.655	233.695	235.654
		Avg	**66.407**	68.998	66.679	68.511	151.768	144.389	142.179	236.277	231.274	233.031
D_6	300	Min	**20.300**	21.324	20.600	21.816	35.892	34.222	38.002	38.696	37.067	40.703
		Max	**20.606**	25.610	21.514	23.305	39.573	34.525	38.280	43.352	38.837	41.195
		Avg	**20.412**	23.303	21.077	22.851	37.991	34.350	38.112	40.338	38.347	40.964
	900	Min	55.891	**55.881**	56.112	57.728	107.681	104.625	108.412	129.248	127.836	130.677
		Max	56.877	60.508	**56.811**	59.122	109.710	106.115	110.034	135.833	131.669	132.884
		Avg	**56.185**	58.464	56.507	58.322	108.418	105.422	108.966	131.469	129.420	131.899
	1500	Min	**92.022**	93.339	92.170	93.697	188.914	188.526	190.508	251.495	249.411	250.938
		Max	**93.022**	98.200	93.025	94.655	194.341	191.091	192.220	264.148	253.773	256.640
		Avg	**92.032**	95.466	92.498	94.106	191.243	189.838	191.406	255.573	251.396	253.024

ผลการทดลองกรณีเปรียบเทียบการจัดตารางกรณีสภาพแวดล้อมภายในเครื่องเสมือนแตกต่างกัน เมื่อทำการทดสอบโดยใช้ชุดข้อมูลแบบสุ่มที่มีการกระจายข้อมูลแบบแจกแจงปกติ แสดงผลการทดลองได้ดังตารางที่ 4.14 ผลการทดลองแสดงให้เห็นว่า เมื่อชุดข้อมูลที่มีช่วงขนาดของ

ข้อมูลที่แตกต่างกัน การจัดตารางงานโดยใช้ HABC_LJF ยังคงมีประสิทธิภาพการทำงานที่ดีกว่าวิธีการอื่นที่นำมาเปรียบเทียบเช่นเดียวกับกรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในเหมือนกัน ยกเว้นกรณีชุดข้อมูลที่ D_4 และ D_6 เนื่องจากกรณีทำการทดสอบกับชุดข้อมูล D_4 เมื่อทดสอบกับงานจำนวน 900 งาน พบว่า ACO_LJF ให้ค่าเวลาทำงานรวมมากที่สุดที่มีค่าน้อยที่สุดเมื่อเปรียบเทียบกับ HABC_LJF, PSO และ IPSO ในขณะเดียวกัน เมื่อทำการทดสอบกับชุดข้อมูล D_6 จำนวน 900 งาน พบว่า ACO_LJF ให้ค่าเวลาทำงานรวมที่น้อยที่สุด และ ACO_SJF ให้ค่าเวลาทำงานรวมมากที่สุดที่มีค่าน้อยที่สุดเมื่อเปรียบเทียบกับอัลกอริทึมอื่นที่นำมาเปรียบเทียบ แต่อย่างไรก็ตามพบว่า ค่าที่ได้นั้นไม่แตกต่างจากการใช้วิธีการจัดตารางโดยใช้ HABC_LJF มากนัก หากเมื่อพิจารณาจากเวลาทำงานรวมในทุกกรณีแล้วพบว่า HABC_LJF ยังคงมีประสิทธิภาพที่ดีกว่าอัลกอริทึมอื่นที่นำมาเปรียบเทียบ

กรณีที่ 3: ชุดข้อมูลที่มีการกระจายข้อมูลแบบเบ้ขวา

การประเมินประสิทธิภาพในการทำงานเปรียบเทียบกับจำนวนงานที่มีการเปลี่ยนแปลง เมื่อจำนวนเครื่องคอมพิวเตอร์เสมือนมีค่าคงที่ ในการทดลองใช้ชุดข้อมูลที่แสดงในตารางที่ 4.1 โดยใช้ชุดข้อมูล $D_7 – D_9$ คือชุดข้อมูลแบบสุ่มที่มีการกระจายข้อมูลแบบเบ้ขวา ซึ่งแต่ละชุดข้อมูลนั้นมีการกำหนดช่วงขนาดของงานที่แตกต่างกัน เพื่อทำการเปรียบเทียบความสามารถในการจัดตารางงานโดยใช้วิธี HABC_LJF เปรียบเทียบกับวิธีการจัดตารางงานโดยใช้ ACO, PSO และ IPSO ในกรณีที่สภาพแวดล้อมภายในเครื่องเสมือนเหมือนกันและแตกต่างกันทั้งหมด ผลการทดลองแสดงได้ดังตารางที่ 4.15 และตารางที่ 4.16

ตารางที่ 4.15 เวลาการทำงานรวมของชุดข้อมูลที่ $D_7 – D_9$ เมื่อเปรียบเทียบระหว่าง HABC, ACO, PSO และ IPSO (Homogenous)

Dataset	Task	Variable	HABC_LJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_FCFS	IPSO_LJF	IPSO_SJF
D_7	300	Min	**8.173**	9.523	8.788	9.600	25.399	24.606	25.048	28.675	28.359	28.078
		Max	**8.248**	10.350	9.068	10.213	25.589	24.846	26.086	29.708	30.863	28.928
		Avg	**8.211**	9.951	8.950	9.904	25.524	24.696	25.554	29.090	29.149	28.515
	900	Min	**23.742**	25.072	24.179	25.155	83.196	81.680	81.820	106.287	104.397	105.338
		Max	**24.130**	26.153	24.727	26.494	83.696	82.620	83.303	110.536	106.107	109.230
		Avg	**23.874**	25.520	24.384	25.640	83.526	82.159	82.205	107.879	105.350	106.786
	1500	Min	**39.575**	41.225	40.088	41.291	151.844	149.901	150.912	227.688	211.238	214.100
		Max	**39.866**	42.121	40.807	42.272	152.875	152.081	152.960	239.956	228.046	230.281
		Avg	**39.679**	41.569	40.433	41.866	152.226	151.149	152.024	232.735	216.979	224.078
D_8	300	Min	**14.604**	16.839	16.258	17.553	32.596	32.379	23.083	37.881	37.521	36.366
		Max	**14.622**	19.080	16.541	18.576	33.470	34.339	23.167	39.898	38.762	37.016
		Avg	**14.613**	17.792	16.406	17.829	32.839	33.012	23.097	38.637	37.980	36.641
	900	Min	**44.195**	46.969	45.842	47.344	107.407	106.832	67.939	132.910	131.491	130.931
		Max	**44.310**	48.608	47.310	49.409	121.232	107.571	68.224	139.303	135.710	132.904
		Avg	**44.234**	47.815	46.140	48.094	115.012	107.174	68.018	135.926	133.278	131.848
	1500	Min	**73.733**	77.107	75.205	76.916	193.523	189.848	113.247	248.315	245.764	248.212
		Max	**75.611**	78.529	77.193	78.873	201.949	192.243	113.367	260.620	252.230	253.200
		Avg	**74.095**	77.772	76.394	78.122	197.262	191.025	113.305	253.714	249.900	250.838
D_9	300	Min	**20.098**	23.697	22.668	23.842	38.213	36.604	37.460	43.877	43.646	42.599
		Max	**20.174**	24.954	23.046	24.824	38.548	36.794	37.628	45.133	47.313	43.088
		Avg	**20.126**	24.223	22.818	24.297	38.421	36.713	37.548	44.439	44.296	42.790
	900	Min	**60.023**	63.854	62.451	64.003	120.953	118.471	89.919	145.600	146.591	141.441
		Max	**60.302**	66.557	64.231	67.714	121.867	119.243	97.388	149.600	170.342	161.315
		Avg	**60.181**	64.803	62.795	64.750	121.292	118.856	92.890	146.600	152.863	150.354
	1500	Min	**99.961**	104.231	102.648	104.687	214.429	213.021	177.504	278.953	268.821	267.217
		Max	**100.364**	106.827	105.310	107.602	217.676	244.700	203.813	309.324	323.782	283.917
		Avg	**100.160**	105.578	104.546	106.389	215.562	224.760	190.212	291.425	286.531	274.623

ตารางที่ 4.16 เวลาการทำงานรวมของชุดข้อมูลที่ D_7 – D_9 เมื่อเปรียบเทียบระหว่าง HABC, ACO, PSO และ IPSO (Heterogenous)

Dataset	Task	Variable	HABC_LJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_FCFS	IPSO_LJF	IPSO_SJF
D_7	300	Min	**7.157**	7.739	7.289	8.396	24.196	22.050	24.566	26.869	26.160	26.898
		Max	**7.173**	8.962	7.622	9.255	27.089	22.625	24.966	35.178	26.884	27.933
		Avg	**7.160**	8.500	7.444	8.802	25.044	22.279	24.726	29.741	26.446	27.167
	900	Min	**19.203**	20.137	19.238	20.553	78.132	72.733	76.368	98.286	97.163	98.379
		Max	**19.441**	21.740	19.828	21.508	107.029	73.530	77.370	104.773	99.165	100.482
		Avg	**19.333**	20.989	19.542	20.926	86.726	73.098	76.781	100.391	98.151	99.511
	1500	Min	**31.908**	32.734	31.960	33.023	134.903	137.902	138.639	202.683	199.200	200.864
		Max	**32.326**	34.789	32.494	35.537	150.393	140.233	141.081	214.879	204.439	206.623
		Avg	**32.045**	33.630	32.307	33.614	139.391	139.147	139.769	206.454	201.669	203.649
D_8	300	Min	**12.967**	14.688	13.103	15.059	29.644	27.963	31.369	32.367	31.856	33.968
		Max	**13.041**	16.495	14.136	16.599	32.058	28.574	31.682	34.567	32.874	37.191
		Avg	**12.994**	15.451	13.617	15.689	30.879	28.262	31.464	33.658	32.236	34.868
	900	Min	**36.784**	37.921	37.188	38.609	90.470	88.877	95.021	113.357	111.705	113.662
		Max	**37.245**	41.936	37.716	40.397	93.650	91.074	113.524	117.442	113.933	116.047
		Avg	**36.984**	39.784	37.400	39.208	91.836	89.756	102.922	115.047	112.727	114.897
	1500	Min	**60.618**	61.338	60.874	62.194	163.189	162.642	165.601	225.678	223.370	225.126
		Max	**60.920**	66.208	61.390	63.100	167.998	165.742	177.877	235.020	228.155	230.726
		Avg	**60.728**	63.599	61.113	62.608	166.105	163.988	171.501	229.573	225.446	227.433
D_9	300	Min	**18.040**	20.069	18.699	20.497	35.894	32.684	36.773	38.833	36.874	39.586
		Max	**18.094**	21.962	19.877	21.893	37.445	33.098	37.211	40.411	37.782	40.017
		Avg	**18.051**	20.827	19.253	21.114	36.423	32.913	36.958	39.699	37.208	39.750
	900	Min	50.580	**50.090**	50.979	52.478	103.374	99.892	103.903	126.204	124.505	127.557
		Max	**50.879**	54.592	51.889	53.356	104.394	101.280	104.900	131.245	127.389	130.504
		Avg	**50.670**	52.765	51.231	52.859	103.951	100.736	104.504	128.347	125.549	128.516
	1500	Min	**82.976**	83.644	83.184	84.285	182.045	180.425	182.410	243.361	241.335	243.867
		Max	84.526	90.443	**83.923**	85.677	187.234	184.087	184.573	255.820	246.441	249.162
		Avg	**83.412**	85.631	83.573	84.847	183.959	181.961	183.547	247.869	243.910	245.499

จากตารางที่ 4.15 และตารางที่ 4.16 แสดงให้เห็นว่าจากการใช้ชุดข้อมูลที่ D_7 – D_9 ที่มีการกระจายข้อมูลแบบเบ้ขวา ทั้งในกรณีที่สภาพแวดล้อมภายในเครื่องเสมือนเหมือนกันและแตกต่างกันตามลำดับ ผลจากการทดลองแสดงให้เห็นว่า HABC_LJF มีประสิทธิภาพในการทำงานที่เหนือกว่าวิธีการอื่นที่นำมาเปรียบเทียบ เช่นเดียวกับกรณีอื่นๆ

กรณีที่สภาพแวดล้อมภายในเครื่องคอมพิวเตอร์เสมือนแตกต่างกัน การจัดตารางงานโดยใช้วิธี HABC_LJF ยังคงมีประสิทธิภาพการทำงานที่เหนือกว่าการจัดตารางงานโดยใช้วิธี ACO, PSO และ IPSO ยกเว้นแต่เมื่อทำการทดลองโดยใช้ชุดข้อมูลที่ D_9 จำนวนงาน 900 และ 1500 งาน ซึ่งเมื่อมีปริมาณงานในระบบ 900 งาน ACO_FCFS ให้ค่าเวลาในการทำงานรวมที่น้อยที่สุด และเมื่อมีปริมาณงาน 1500 งานในระบบ พบว่า ACO_LJF ให้ค่าเวลาทำงานรวมมากที่สุดที่มีค่าน้อยที่สุดเมื่อเปรียบเทียบกับวิธีการอื่นที่นำมาเปรียบเทียบ แต่อย่างไรก็ตาม เมื่อพิจารณาจากทั้ง 2 กรณีที่เกิดขึ้นนั้น ค่าที่ได้นั้นไม่แตกต่างจากการใช้ HABC_LJF ซึ่งหากพิจารณาโดยรวมแล้วพบว่า HABC_LJF ยังมีประสิทธิภาพการทำงานที่ดีที่สุดเมื่อเปรียบเทียบกับ ACO, PSO และ IPSO

กรณีที่ 4: ชุดข้อมูลที่มีการกระจายข้อมูลแบบเบ้ซ้าย

การประเมินประสิทธิภาพในการทำงานเปรียบเทียบกับจำนวนงานที่มีการเปลี่ยนแปลง เมื่อจำนวนเครื่องคอมพิวเตอร์เสมือนมีค่าคงที่ ในการทดลองใช้ชุดข้อมูลที่แสดงในตารางที่ 4.1 โดยใช้ชุดข้อมูล D_{10} – D_{12} คือชุดข้อมูลแบบสุ่มที่มีการกระจายข้อมูลแบบเบ้ซ้าย ซึ่งแต่ละชุดข้อมูลนั้นมีการกำหนดช่วงขนาดของงานที่แตกต่างกัน เพื่อทำการเปรียบเทียบความสามารถในการจัดตารางงานของ HABC_LJF algorithm เปรียบเทียบกับ ACO, PSO และ IPSO ในกรณีที่

สภาพแวดล้อมภายในเครื่องเสมือนเหมือนกันและแตกต่างกันทั้งหมด ผลการทดลองแสดงได้ดังตารางที่ 4.17 และตารางที่ 4.18

ตารางที่ 4.17 เวลาการทำงานรวมของชุดข้อมูลที่ D_{10} – D_{12} เมื่อเปรียบเทียบระหว่าง HABC, ACO, PSO และ IPSO (Homogenous)

Dataset	Task	Variable	HABC_LJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_FCFS	IPSO_LJF	IPSO_SJF
D_{10}	300	Min	**9.840**	11.152	10.554	11.320	31.987	27.146	28.775	29.970	29.706	30.153
		Max	**9.943**	12.283	10.891	12.288	36.282	28.566	29.783	34.790	31.869	32.929
		Avg	**9.857**	11.583	10.739	11.773	34.396	28.004	29.273	31.175	30.264	31.182
	900	Min	**29.170**	30.798	29.866	30.870	96.423	89.398	91.702	110.496	108.828	109.847
		Max	**29.390**	31.637	30.393	31.946	101.850	97.457	107.651	114.164	112.227	125.505
		Avg	**29.284**	31.287	30.171	31.318	98.688	93.432	99.153	111.847	110.056	114.298
	1500	Min	**48.753**	50.840	49.349	51.067	168.387	170.806	171.145	236.442	216.480	217.155
		Max	**49.059**	51.677	50.318	51.767	175.637	180.935	180.160	261.561	245.113	256.378
		Avg	**48.904**	51.238	49.811	51.375	172.856	175.650	175.458	246.532	225.088	224.397
D_{11}	300	Min	**17.111**	19.555	18.880	20.020	37.320	34.579	34.573	38.583	37.057	37.894
		Max	**17.142**	20.752	19.158	21.054	38.739	40.478	35.841	40.244	41.899	38.477
		Avg	**17.124**	20.208	19.026	20.398	38.062	36.998	35.571	39.567	39.499	38.056
	900	Min	**51.537**	54.776	53.445	54.654	115.854	109.765	111.203	131.665	129.210	131.846
		Max	**51.832**	56.250	54.977	57.353	117.708	110.733	112.638	149.719	146.301	153.488
		Avg	**51.637**	55.537	53.830	55.320	116.586	110.284	111.802	136.028	134.417	138.309
	1500	Min	**85.919**	89.375	88.298	89.697	215.555	196.344	199.781	262.457	261.945	261.034
		Max	**86.397**	90.958	89.611	91.721	238.502	201.305	200.884	324.247	278.191	284.655
		Avg	**86.059**	90.182	89.136	90.462	224.720	199.123	200.360	332.678	265.998	269.802
D_{12}	300	Min	**22.244**	25.945	24.950	26.195	40.482	40.616	39.496	46.954	45.572	44.392
		Max	**22.371**	27.360	25.323	27.377	40.875	43.034	40.046	56.344	50.751	50.231
		Avg	**22.274**	26.432	25.186	26.673	40.662	42.176	39.677	49.659	47.707	46.474
	900	Min	**67.354**	70.989	70.207	71.414	129.654	127.078	126.089	149.600	149.927	153.897
		Max	**67.617**	73.371	72.689	75.204	130.476	135.066	128.023	178.262	182.307	171.051
		Avg	**67.514**	72.282	70.634	72.271	130.069	131.126	126.724	159.679	158.755	159.720
	1500	Min	**112.013**	117.006	115.147	116.653	239.050	222.857	223.367	287.868	285.495	286.239
		Max	**112.415**	118.822	117.796	119.924	256.957	227.268	226.719	298.553	290.658	291.120
		Avg	**112.265**	117.840	117.247	118.586	246.514	224.318	225.411	292.954	288.298	288.380

ตารางที่ 4.18 เวลาการทำงานรวมของชุดข้อมูลที่ D_{10} – D_{12} เมื่อเปรียบเทียบระหว่าง HABC, ACO, PSO และ IPSO (Heterogenous)

Dataset	Task	Variable	HABC_LJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_FCFS	IPSO_LJF	IPSO_SJF
D_{10}	300	Min	**8.570**	9.211	8.997	9.416	25.328	23.257	25.216	34.267	33.909	34.000
		Max	**8.594**	10.540	9.871	10.670	26.559	24.303	25.566	37.286	35.337	38.069
		Avg	**8.579**	9.945	9.332	10.127	25.791	23.569	25.368	35.458	34.199	36.175
	900	Min	**23.705**	24.778	24.090	25.086	79.953	77.752	79.527	134.436	122.920	106.875
		Max	**23.857**	26.676	24.448	26.233	80.804	79.033	80.539	149.669	136.827	132.263
		Avg	**23.765**	25.765	24.271	25.683	80.418	78.358	80.107	137.860	132.622	121.142
	1500	Min	**39.475**	40.010	39.631	40.767	145.304	143.807	145.584	208.322	204.563	207.098
		Max	**40.476**	42.386	40.197	43.376	148.738	146.613	150.235	217.963	210.574	212.015
		Avg	**39.875**	41.147	39.977	41.198	146.693	145.407	147.045	211.902	207.255	209.350
D_{11}	300	Min	**15.339**	16.487	15.730	16.377	31.885	29.381	32.504	37.556	36.849	37.450
		Max	**15.431**	18.755	16.581	18.339	34.454	29.743	33.134	47.160	38.005	40.015
		Avg	**15.369**	17.386	16.103	17.536	32.801	29.597	32.665	39.769	37.318	38.043
	900	Min	**42.742**	43.518	43.238	44.708	96.971	94.199	97.169	130.707	130.652	131.525
		Max	43.858	47.344	**43.846**	46.198	98.603	95.061	97.939	137.456	135.611	135.364
		Avg	**42.869**	45.593	43.540	45.321	97.502	94.616	97.492	133.879	131.903	133.123
	1500	Min	**70.269**	71.839	70.801	72.098	172.093	170.612	171.553	240.058	295.388	304.815
		Max	**70.547**	75.895	72.058	73.386	175.863	172.934	173.107	319.583	314.065	314.763
		Avg	**70.383**	73.419	71.294	72.809	173.570	171.435	172.327	303.582	306.437	308.351
D_{12}	300	Min	**20.479**	22.064	20.784	22.039	37.282	34.274	38.110	39.859	36.214	40.114
		Max	**20.554**	24.029	21.434	23.901	39.306	34.794	38.501	42.938	39.384	46.617
		Avg	**20.504**	22.897	21.048	22.927	37.732	34.544	38.310	40.846	37.953	42.435
	900	Min	**56.622**	56.887	56.785	57.182	108.605	105.402	109.153	129.440	126.990	130.697
		Max	**56.770**	63.778	57.721	60.232	110.586	106.888	110.214	135.108	129.848	132.790
		Avg	**56.673**	59.780	57.240	58.976	109.669	106.044	109.677	131.548	128.587	131.451
	1500	Min	**92.805**	94.120	93.016	94.593	190.609	189.282	190.582	250.071	246.614	340.863
		Max	**93.225**	98.253	93.849	95.619	194.868	192.231	192.333	263.459	350.374	356.086
		Avg	**92.951**	96.197	93.379	95.041	192.957	190.811	191.589	256.314	297.956	347.719

ผลการทดลองในตารางที่ 4.17 แสดงให้เห็นว่าการจัดตารางงานเข้าทำงานโดยใช้วิธี HABC_LJF มีประสิทธิภาพการทำงานที่ดีกว่าการจัดตารางงานโดยใช้วิธี ACO, PSO และ IPSO

ในกรณีที่เครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในเหมือนกัน ซึ่งให้ผลลัพธ์ไปในทางเดียวกันกับกรณีอื่นๆ

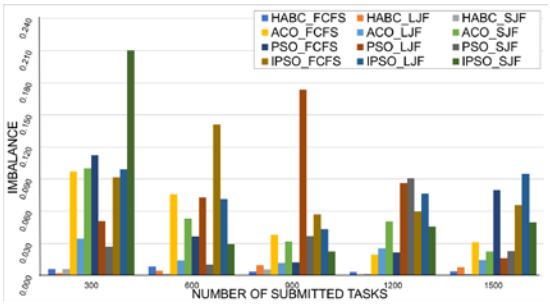
กรณีที่เครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในแตกต่างกัน ผลการทดลองแสดงได้ดังตารางที่ 4.18 พบว่าการจัดตารางงานเข้าทำงานโดยใช้วิธี HABC_LJF มีประสิทธิภาพการทำงานที่ดีกว่าวิธีการที่นำมาเปรียบเทียบ ยกเว้นเพียงกรณีที่ทำการทดลองโดยใช้ชุดข้อมูล D_{11} เมื่อทดสอบกับงานจำนวน 900 งาน พบว่า ACO_LJF ให้ค่าเวลาทำงานรวมที่น้อยที่สุด ซึ่งหากทำการเปรียบเทียบกับค่าของ HABC_LJF จะเห็นได้ว่า ค่าทั้งสองไม่แตกต่างกันมาก และหากพิจารณาในทุกกรณีแล้วจะเห็นได้ว่า HABC_LJF ยังคงมีประสิทธิภาพในการทำงานที่ดีว่าวิธีการอื่น

4.5.3.2 ผลการทดสอบประสิทธิภาพในแง่ของค่าความไม่สมดุลของการกระจายข้อมูล

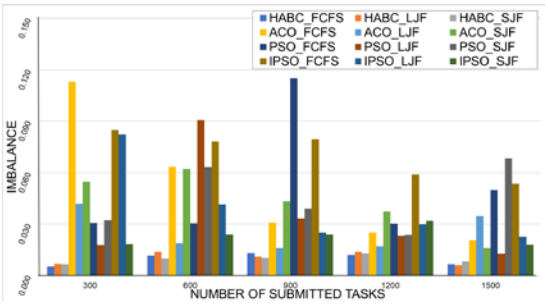
ในส่วนนี้ทำการทดสอบประสิทธิภาพของ HABC ในแง่มุมของค่าความไม่สมดุล (DI) ของการกระจายงาน ซึ่งหากเมื่อมีการกระจายงานแบบไม่สมดุลนั้น ส่งผลให้เครื่องคอมพิวเตอร์เสมือนบางเครื่องอาจทำงานไม่เหมาะสมได้ ในการทดลองใช้ชุดข้อมูลที่แสดงในตารางที่ 4.1 ในการทดสอบได้แบ่งการประเมินประสิทธิภาพออกเป็น 4 กรณี ซึ่งพิจารณาจากประเภทของชุดข้อมูลที่นำมาใช้ในการทดสอบ ในการทดสอบความสามารถในการทำงานเปรียบเทียบกับจำนวนงานที่มีการเปลี่ยนแปลง โดยกำหนดให้จำนวนงานเพิ่มขึ้นครั้งละ 100 งาน เริ่มจาก 100 – 1500 งาน ในการทดสอบประสิทธิภาพนั้นยังคงแบ่งแยกการทดลองตามสภาพแวดล้อมของเครื่องคอมพิวเตอร์เสมือน ผลที่ได้จากการทดสอบนั้นเกิดจากการทดลองซ้ำจำนวน 20 ครั้งสำหรับแต่ละวิธีการ ซึ่งผลที่ได้จากการทดสอบประสิทธิภาพของค่าความไม่สมดุลของการกระจายข้อมูลในการประมวลผลแบบกลุ่มเมฆ กรณีที่เครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในเหมือนกันและแตกต่างกัน แสดงได้ดังรูปที่ 4.7 และ 4.8 ตามลำดับ

จากผลการทดลองในรูปที่ 4.7 ได้ประเมินประสิทธิภาพของค่าความไม่สมดุลของการกระจายข้อมูล โดยใช้ชุดข้อมูลที่ D_1, D_4 , D_7 และ D_{10} แสดงให้เห็นว่าการจัดตารางงานโดยใช้ขั้นตอนอาณานิคมผึ้งเทียมควบคู่กับการจัดลำดับงานแบบฮิวริสติก (HABC_FCFS, HABC_LJF, HABC_SJF) มีค่าความไม่สมดุลของการกระจายข้อมูลเฉลี่ยที่น้อยกว่าอัลกอริทึม ACO, PSO และ IPSO ในกรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในที่เหมือนกัน

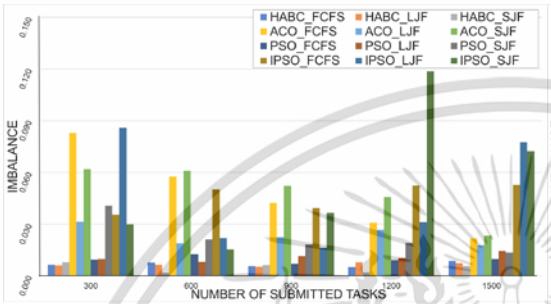
ในทิศทางเดียวกัน ผลการทดลองในรูปที่ 4.8 แสดงให้เห็นว่า ในกรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในที่แตกต่างกัน การจัดตารางงานโดยใช้ขั้นตอนอาณานิคมผึ้งเทียมควบคู่กับการจัดลำดับงานแบบฮิวริสติก (HABC_FCFS, HABC_LJF, HABC_SJF) มีค่าความไม่สมดุลของการกระจายข้อมูลเฉลี่ยที่น้อยกว่าอัลกอริทึม ACO, PSO และ IPSO เช่นเดียวกับกรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในที่เหมือนกัน



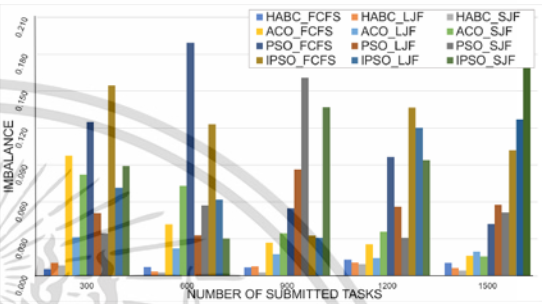
(ก) ชุดข้อมูลแบบสุ่มอิสระ



(ข) ชุดข้อมูลแบบการแจกแจงปกติ

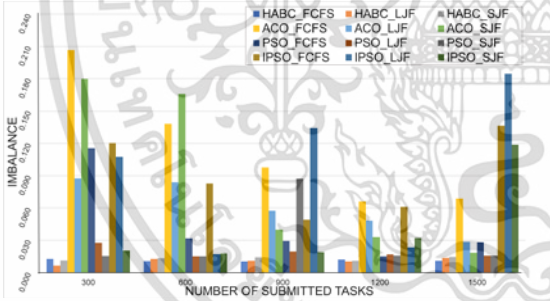


(ค) ชุดข้อมูลแบบการกระจายข้อมูลแบบเบ้ซ้าย

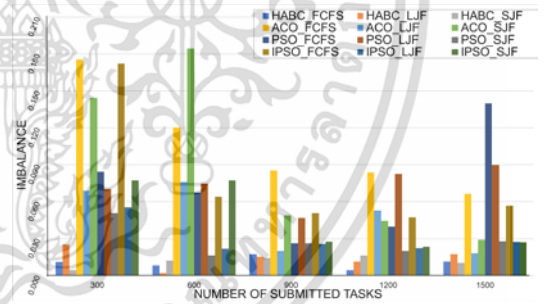


(ง) ชุดข้อมูลแบบการกระจายข้อมูลแบบเบ้ขวา

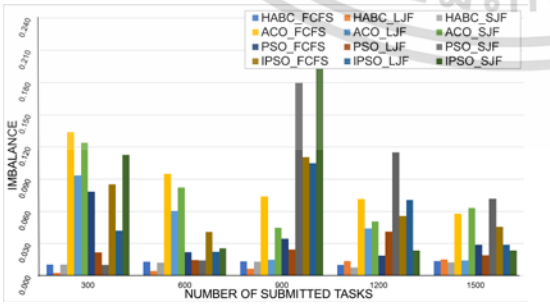
รูปที่ 4.7 ค่าความไม่สมดุลของการกระจายข้อมูล กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในเหมือนกัน



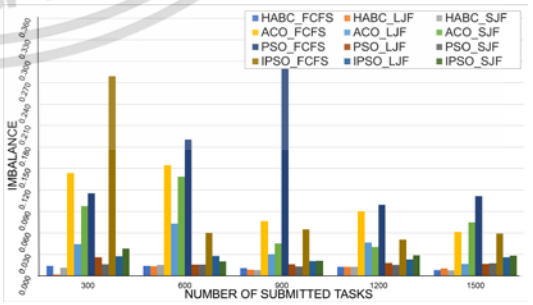
(ก) ชุดข้อมูลแบบสุ่มอิสระ



(ข) ชุดข้อมูลแบบการแจกแจงปกติ



(ค) ชุดข้อมูลแบบการกระจายข้อมูลแบบเบ้ซ้าย



(ง) ชุดข้อมูลแบบการกระจายข้อมูลแบบเบ้ขวา

รูปที่ 4.8 ค่าความไม่สมดุลของการกระจายข้อมูล กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในแตกต่างกัน

4.5.3.3 ผลการทดสอบประสิทธิภาพในส่วนของค่าเบี่ยงเบนมาตรฐาน

การทดสอบประสิทธิภาพของ HABC ในแง่มุมของค่าเบี่ยงเบนมาตรฐาน (S.D) เพื่อเป็นการทดสอบว่ามีการกระจายงานไปยังเครื่องคอมพิวเตอร์เสมือนอย่างสมดุล ในการทดลองได้เลือกใช้ชุดข้อมูล D_1 จากตารางที่ 4.1 ในการทดสอบความสามารถในการทำงานเปรียบเทียบกับจำนวนงานที่มีการเปลี่ยนแปลง โดยกำหนดให้จำนวนงานเพิ่มขึ้นครั้งละ 100 งาน เริ่มจาก 100 – 1500 งาน ในการทดสอบประสิทธิภาพนั้นยังคงแบ่งแยกการทดลองตามสภาพแวดล้อมของเครื่องคอมพิวเตอร์เสมือน ผลที่ได้จากการทดสอบนั้นเกิดจากการทดลองซ้ำจำนวน 20 ครั้งสำหรับแต่ละวิธีการ ซึ่งผลที่ได้จากการทดสอบประสิทธิภาพของค่าความไม่สมดุลของการกระจายข้อมูลในการประมวลผลแบบกลุ่มเมฆ กรณีที่เครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในเหมือนกันและแตกต่างกัน แสดงได้ดังตารางที่ 4.19 และ 4.20 ตามลำดับ

ตารางที่ 4.19 ค่าส่วนเบี่ยงเบนมาตรฐาน กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในเหมือนกัน

Number of Tasks	HABC_ FCFS	HABC_ LJF	HABC_ SJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_ FCFS	IPSO_LJF	IPSO_SJF
300	0.12	0.06	0.09	1.09	1.01	1.12	0.51	0.38	0.46	0.21	0.14	0.16
600	0.11	0.05	0.08	2.07	2.03	2.09	0.51	0.38	0.45	0.21	0.14	0.15
900	0.09	0.04	0.07	3.02	2.99	3.04	0.48	0.37	0.44	0.18	0.13	0.14
1200	0.08	0.04	0.08	4.08	4.06	4.1	0.49	0.37	0.45	0.19	0.13	0.15
1500	0.09	0.07	0.09	5.05	5.03	5.06	0.52	0.36	0.44	0.22	0.12	0.14

ตารางที่ 4.20 ค่าส่วนเบี่ยงเบนมาตรฐาน กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในแตกต่างกัน

Number of Tasks	HABC_ FCFS	HABC_ LJF	HABC_ SJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_ FCFS	IPSO_LJF	IPSO_SJF
300	0.16	0.12	0.14	0.71	0.63	0.74	0.86	0.52	1.18	0.41	0.22	0.48
600	0.17	0.09	0.15	1.32	1.27	1.33	0.89	0.55	1.29	0.44	0.25	0.59
900	0.15	0.12	0.13	1.93	1.89	1.9	0.94	0.52	1.26	0.49	0.22	0.56
1200	0.17	0.13	0.12	2.57	2.56	2.59	0.87	0.53	1.03	0.42	0.23	0.33
1500	0.16	0.11	0.13	3.19	3.17	3.2	0.82	0.65	0.73	0.37	0.25	0.23

จากตารางที่ 4.19 แสดงค่าส่วนเบี่ยงเบนมาตรฐานของอัลกอริทึมทั้งหมดที่นำมาใช้ในการทดสอบประสิทธิภาพ โดยใช้ชุดข้อมูล D_1 ผลจากการทดสอบประสิทธิภาพพบว่าการจัดตารางงานโดยใช้ขั้นตอนอาณานิคมผึ้งเทียมควบคู่กับการจัดลำดับงานแบบฮิวริสติก (HABC_FCFS, HABC_LJF และ HABC_SJF) มีค่าส่วนเบี่ยงเบนมาตรฐานที่น้อยกว่าการจัดตารางงานโดยใช้อัลกอริทึม ACO, PSO และ IPSO ในกรณีที่เครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในเหมือนกัน

กรณีที่เครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในแตกต่างกัน ดังแสดงในตารางที่ 4.20 ในการทดลองก่อนหน้าพบว่า ค่าเวลาทำงานรวมและค่าความไม่สมดุลของการกระจายข้อมูลในบางกรณีวิธีการ HABC และ ACO มีค่าแตกต่างกันเล็กน้อย แต่หากพิจารณาในแง่ของการ

เปรียบเทียบประสิทธิภาพโดยใช้ค่าส่วนเบี่ยงเบนมาตรฐานนั้นพบว่า วิธีการ HABC และ ACO นั้นมีค่าที่แตกต่างกันมาก ดังนั้นผลจากการทดสอบประสิทธิภาพพบว่า อัลกอริทึม HABC มีประสิทธิภาพการทำงานที่ดีกว่าอัลกอริทึม ACO, PSO และ IPSO ในการจัดตารางการทำงานในระบบการประมวลผลแบบกลุ่มเมฆ ทั้งในกรณีที่เครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในที่เหมือนกันและแตกต่างกัน อีกทั้งยังช่วยให้เกิดความสมดุลระหว่างทรัพยากรในระบบ และยังช่วยลดเวลาในการทำงานให้น้อยลงด้วย

ซึ่งจากการทดลองทั้งหมดที่ได้นำเสนอนั้นจะใช้กระบวนการจัดตารางงานที่มีลักษณะคล้ายคลึงกันโดยยึดตามวิธีการจัดตารางงานโดยใช้ฮิวริสติก เมื่อทำการเปรียบเทียบแล้วพบว่า อัลกอริทึม HABC สามารถแสดงให้เห็นถึงประสิทธิภาพการทำงานได้ดีกว่าอัลกอริทึม ACO, PSO และ IPSO โดยเฉพาะอย่างยิ่งในกรณีที่ส่งงานขนาดใหญ่เข้าทำงานก่อน (HABC_LJF) ซึ่งไม่เพียงแต่จะช่วยให้ระบบเกิดการกระจายงานที่สมดุลแล้วยังช่วยลดเวลาในการเข้าทำงานในระบบได้อีกด้วย

4.6 ผลการทดลองเพื่อประเมินประสิทธิภาพของการจัดตารางงานโดยใช้ MOABCQ

การตั้งค่าพารามิเตอร์และผลการทดสอบประสิทธิภาพของการจัดตารางงาน เพื่อประเมินประสิทธิภาพของวิธีการจัดตารางงานที่เป็นอิสระต่อกัน (Independent task) ในการประมวลผลแบบกลุ่มเมฆ โดยมุ่งเน้นที่วิธีการจัดตารางงานแบบที่มีหลายวัตถุประสงค์ (Multi-objective optimization scheduling method) โดยใช้ขั้นตอนอาณานิคมผึ้งเทียมควบคู่กับการปรับใช้ขั้นตอนวิธีการเรียนรู้แบบคิว เพื่อช่วยให้การใช้ขั้นตอนอาณานิคมผึ้งเทียมสามารถทำงานได้เร็วขึ้น เรียกอัลกอริทึมนี้ว่า “MOABCQ” ซึ่งอัลกอริทึมที่นำเสนอนี้มีวัตถุประสงค์เพื่อเพิ่มประสิทธิภาพการใช้งานของทรัพยากรของระบบ และให้มีการกระจายงานระหว่างเครื่องคอมพิวเตอร์เสมือนแบบสมดุล (Load balancing) โดยพิจารณาจากเวลารวมทั้งหมดในการทำงาน (Makespan) ค่าใช้จ่ายที่ใช้ในการเข้าทำงานในเครื่องคอมพิวเตอร์เสมือนของงาน และการใช้งานทรัพยากรพร้อมกันน้อยที่สุด (Utilization of resources) เป็นข้อจำกัดของการพิจารณาพร้อมกัน (Optimization problem)

การประเมินประสิทธิภาพในการจัดตารางโดยใช้อัลกอริทึมที่นำเสนอ ถูกนำไปเปรียบเทียบกับวิธีการจัดตารางงานโดยใช้วิธีการทางฮิวริสติกที่เป็นที่รู้จัก เช่น ขั้นตอนการจัดตารางงานโดยใช้ Max-Min [53] ขั้นตอนวิธีการจัดตารางงานแบบมาก่อนได้ก่อน (FCFS algorithm) ขั้นตอนการจัดตารางงานแบบงานขนาดใหญ่ที่สุดทำก่อน (LJF algorithm) และยิ่งไปกว่านั้น ยังได้ทำการเปรียบเทียบการจัดตารางงานโดยใช้วิธีการทางเมตาฮิวริสติกที่เป็นที่นิยม คือ ขั้นตอนวิธีการหาคำตอบเหมาะสมที่สุดแบบกลุ่มอนุภาค (PSO algorithm) ขั้นตอนวิธีการหาค่าเหมาะสมที่สุดแบบนกกาเหว่า (CS algorithm) อีกทั้งยังได้ทำการเปรียบเทียบกับวิธีที่นำเสนอในการทดลองก่อนหน้า คือ วิธีการจัดตารางงานโดยใช้ขั้นตอนอาณานิคมผึ้งเทียมควบคู่กับการจัดลำดับงานแบบฮิวริสติก (HABC_LJF)

การประเมินประสิทธิภาพในการจัดตารางโดยใช้อัลกอริทึมที่นำเสนอ ประกอบด้วยส่วนต่างๆ ดังนี้ การตั้งค่าพารามิเตอร์สภาพแวดล้อมในการทดลอง การตั้งค่าพารามิเตอร์สำหรับอัลกอริทึมเมตาฮิวริสติกที่ใช้ในการเปรียบเทียบ ชุดข้อมูลที่ใช้ในการทดลอง และผลการทดสอบประสิทธิภาพของวิธีการที่นำเสนอ

4.6.1 การตั้งค่าพารามิเตอร์สภาพแวดล้อมในการทดลอง

การตั้งค่าพารามิเตอร์แบบจำลองเพื่อประเมินประสิทธิภาพของวิธีการที่นำเสนอ (MOABCQ) เมื่อทำการเปรียบเทียบกับการจัดตารางงานโดยใช้วิธีการอื่นๆ ในระบบการประมวลผลแบบกลุ่มเมฆ กรณีเครื่องคอมพิวเตอร์เสมือนมีสภาพแวดล้อมภายในแตกต่างกัน ซึ่งการทดลองได้ใช้โปรแกรม CloudSim 3.0.3 ในการจำลองสภาพแวดล้อมเสมือนจริงของระบบการประมวลผลแบบกลุ่มเมฆ การตั้งค่าพารามิเตอร์ของแบบจำลองสภาพแวดล้อมเสมือนจริงแสดงได้ดังตารางที่ 4.21

ตารางที่ 4.21 ค่าพารามิเตอร์ของแบบจำลองสภาพแวดล้อมในการทดลอง

Type	Parameter	Value
Host	Number of Host	20
	MIPS	177,730
	Bandwidth	10 GB/s
	Storage	2 TB
	RAM	16 GB
Data Center	Number of Data Center	1
	VmScheduler	Time Shared
	Cost per Memory	0.1 - 1.0
	Cost per Storage	0.1 - 1.0
	VM Monitor	Xen
Cloudlet (Task)	Length of Tasks	1k – 900k
	Number of Tasks	200-1,000
Virtual Machine (VM)	Number of VMs	5 - 100
	Processor speed	3,500 - 100,000 MIPS
	Memory	1 – 4 GB
	Bandwidth	1,000 – 10,000
	Cost per Memory	0.1 - 1.0

ตารางที่ 4.21 ค่าพารามิเตอร์ของแบบจำลองสภาพแวดล้อมในการทดลอง (ต่อ)

Type	Parameter	Value
Virtual Machine (VM)	Cost per Storage	0.1 - 1.0
	Cloudlet Scheduler	Time Shared
	Number of PEs	1
	VM Monitor	Xen

4.6.2 การตั้งค่าพารามิเตอร์สำหรับอัลกอริทึมเมตาฮิวริสติกที่ใช้ในการเปรียบเทียบ

ในการทดลองมีการกำหนดค่าพารามิเตอร์ของประชากรและเงื่อนไขต่างๆ โดยอ้างอิงพารามิเตอร์ตามบทความที่เกี่ยวข้อง ในการประเมินประสิทธิภาพของวิธีการที่นำเสนอโดยทำการเปรียบเทียบกับขั้นตอนวิธีอาณานิคมผึ้งเทียมควบคู่กับการจัดลำดับงานแบบฮิวริสติก (HABC) ขั้นตอนวิธีการหาคำตอบเหมาะสมที่สุดแบบกลุ่มอนุภาค (PSO) [77] และขั้นตอนวิธีการหาค่าเหมาะสมที่สุดแบบนกกาเหว่า (CS) [66] ทั้งนี้เป็นที่ทราบกันดีว่าพารามิเตอร์เหล่านี้มีผลอย่างมากต่อประสิทธิภาพการทำงานของอัลกอริทึม ทั้งนี้การกำหนดพารามิเตอร์จะแตกต่างกันไปตามขนาดและลักษณะของปัญหา ในการทดลองได้กำหนดพารามิเตอร์ของผึ้งที่เหมาะสมที่สุดสำหรับอัลกอริทึมที่นำเสนอ ดังที่ได้เสนอผลการทดลองในหัวข้อที่ 4.4 การตั้งค่าพารามิเตอร์ของอัลกอริทึมเมตาฮิวริสติกที่ใช้ในการเปรียบเทียบ แสดงได้ดังตารางที่ 4.22

ตารางที่ 4.22 ค่าพารามิเตอร์ของอัลกอริทึมเมตาฮิวริสติกที่ใช้ในการเปรียบเทียบ

Algorithms	Parameter	Values
HABC [61]	Number of population (n)	960
	Number of sites selected out of n visited sites (m)	96
	Number of best sites (e)	1
	Number of Employed bees	192
	Number of Onlooker bees	768
	Maximum iterations	1000
MOABCQ [78]	Number of population (n)	960
	Number of sites selected out of n visited sites (m)	96
	Number of best sites (e)	1
	Number of Employed bees	192
	Number of Onlooker bees	768
	Maximum iterations	1000

ตารางที่ 4.22 ค่าพารามิเตอร์ของอัลกอริทึมเมตาฮิวริสติกที่ใช้ในการเปรียบเทียบ (ต่อ)

Algorithms	Parameter	Values
MOPSO [77]	Number of particle size	100
	Weight (w_{min}, w_{max})	0.1, 0.9
	Self-recognition coefficients (c_1, c_2)	1.49445
	Maximum iterations	1000
MOCS [66]	Number of population size	20
	Abandon probability (P_a)	0.25
	Step size (λ)	0.01, 1
	Maximum iterations	1000

4.6.3 ชุดข้อมูลที่ใช้ในการทดลอง

ในการประเมินประสิทธิภาพการจัดตารางงานที่นำเสนอ ได้ใช้ชุดข้อมูลทั้งหมด 3 ชุดที่แตกต่างกัน ดังนี้ 1) ชุดข้อมูลสุ่มแบบอิสระ 2) ชุดข้อมูล Google Cloud Jobs (GoCJ) [11] และ 3) ชุดข้อมูล Synthetic workload [12] มีรายละเอียดดังนี้

1) ชุดข้อมูลสุ่มแบบอิสระ คือชุดข้อมูลที่ถูกสร้างขึ้นแบบสุ่ม โดยมีขนาดของข้อมูลอยู่ระหว่าง 1k – 70k Million Instructions (MI) ซึ่งประกอบด้วยงานจำนวน 1000 งาน แต่ละงานประกอบด้วย ขนาดของงาน (Task size) จำนวน CPU ที่มีการร้องขอ จำนวน RAM ที่มีการร้องขอ เป็นต้น

2) ชุดข้อมูล Google Cloud Jobs (GoCJ dataset) [11] ถือเป็นชุดข้อมูลที่เสมือนจริงของ Google ที่ถูกสร้างขึ้นจากปริมาณการใช้งานจริงที่พบใน Google cluster traces โดยใช้วิธีการเลียนแบบทางสถิติมอนติคาร์โล (Monte Carlo simulation method) ในชุดข้อมูล GoCJ นี้มีการกำหนดขนาดของงานตั้งแต่ 15k-900k MI และมีการแบ่งชุดข้อมูลดังนี้ small size jobs (15k – 55k MI), medium size jobs (59k – 99k MI), large size jobs (101k – 135k MI), extra-large size jobs (150k – 337.5k MI) และ huge size jobs (525k – 900k MI)

3) ชุดข้อมูล Synthetic workload [12] คือชุดข้อมูลที่ถูกสร้างขึ้นด้วยการสุ่ม โดยใช้วิธีการเลียนแบบทางสถิติมอนติคาร์โล ซึ่งประกอบด้วยงานที่มีขนาดของงานที่แตกต่างกันตั้งแต่ 1-45k MI และมีการแบ่งชุดข้อมูลดังนี้ tiny size (1-250 MI), small (800–1200 MI), medium (1800–2500 MI), large (7k–10k MI) และ extra-large (30k–45k MI)

4.6.4 ผลการทดลองเพื่อประเมินประสิทธิภาพในการทำงาน

การประเมินประสิทธิภาพของวิธีการจัดตารางที่นำเสนอ (MOABCQ) โดยใช้ชุดข้อมูลที่กำหนดในการทดสอบ เพื่อเปรียบเทียบผลการทดลองในแง่มุมของเวลาทำงานรวมเฉลี่ย (Makespan) ค่าใช้จ่าย (Cost) อัตราส่วนการใช้ทรัพยากรโดยเฉลี่ย (Average resource utilization ratio:

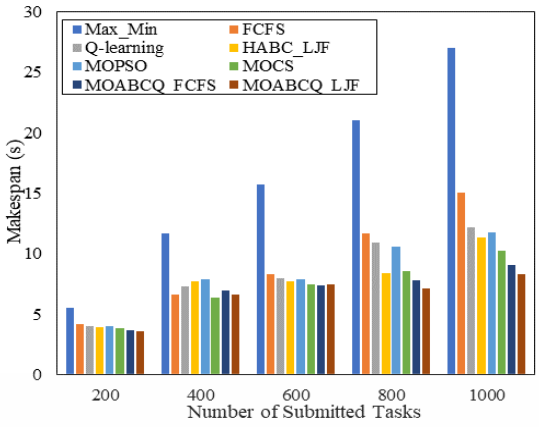
ARUR) ปริมาณงานต่อหน่วยเวลา (Throughput) และค่าความไม่สมดุลของการกระจาย (Degree of imbalance: DI) ในการทดลองเพื่อประเมินประสิทธิภาพของวิธีการที่นำเสนอ (MOABCQ) ทางผู้วิจัยได้นำ MOABCQ รวมเข้ากับวิธีการจัดตารางงานฮิวริสติกแบบลำดับการมาถึงของงาน (FCFS) ซึ่งเรียกว่า “MOABCQ_FCFS” และการจัดตารางงานฮิวริสติกแบบงานขนาดใหญ่ที่สุดเข้าทำงานก่อน (LJF) ซึ่งเรียกว่า “MOABCQ_LJF” อีกทั้งยังทำการเปรียบเทียบกับอัลกอริทึมที่เป็นที่รู้จัก ได้แก่ วิธีการจัดตารางงานแบบลำดับการมาถึงของงาน วิธีการจัดตารางงานแบบ Max-Min ขั้นตอนอาณานิคมผึ้งเทียมที่นำเสนอทำงานร่วมกับการจัดตารางงานโดยพิจารณาจากขนาดของงานที่มีขนาดใหญ่ที่สุดเข้าทำก่อน (HABC_LJF) ขั้นตอนวิธีการเรียนรู้แบบคิว (Q-learning algorithm) ขั้นตอนวิธีการหาคำตอบเหมาะสมที่สุดแบบกลุ่มอนุภาคแบบหลายวัตถุประสงค์ (Multi-objective particle swarm optimization: MOPSO algorithm) และขั้นตอนวิธีการหาค่าเหมาะสมที่สุดของนกกาเหว่าแบบหลายวัตถุประสงค์ (Multi-objective cuckoo search: MOCS algorithm) ผลที่ได้จากการทดสอบนั้นเกิดจากการทดลองซ้ำจำนวน 20 ครั้งสำหรับแต่ละวิธีการ และนำเสนอเป็นค่าเฉลี่ย การประเมินประสิทธิภาพในแต่ละแง่มุมมีรายละเอียดดังต่อไปนี้

4.6.4.1 ผลการทดสอบประสิทธิภาพในกรณีเวลาทำงานรวม

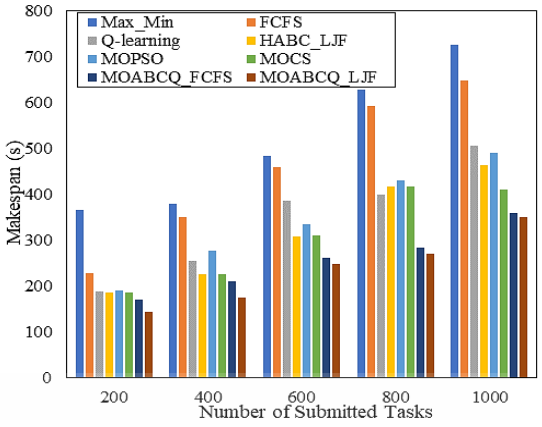
ผลการเปรียบเทียบประสิทธิภาพของวิธีการที่นำเสนอในแง่ของเวลาทำงานรวม ซึ่งในการทดลองได้กำหนดจำนวนเครื่องคอมพิวเตอร์เสมือนเท่ากับ 100 เครื่อง และกำหนดปริมาณงานในการทดลองมีค่าเท่ากับ 200, 400, 600, 800 และ 1000 งาน ในการทดลองได้มีการทดสอบกับชุดข้อมูลจำนวน 3 ชุดที่ได้กล่าวถึงในหัวข้อที่ 4.6.3 ซึ่งผลการทดลองแสดงได้ดังรูปที่ 4.9

จากผลการทดลองในรูปที่ 4.9(ก) แสดงเวลาทำงานรวมโดยทำการเปรียบเทียบขั้นตอนการจัดตารางงานในเครื่องคอมพิวเตอร์เสมือนที่มีสภาพแวดล้อมภายในแตกต่างกัน เมื่อทำการทดสอบโดยใช้ชุดข้อมูลสุ่มแบบอิสระ ผลจากการทดลองแสดงให้เห็นว่า การจัดตารางงานโดยใช้ MOABCQ ให้ค่าเวลาทำงานรวมที่มีค่าน้อยกว่าการจัดตารางงานโดยใช้ Max-Min FCFS, HABC_LJF, Q-learning, MOPSO และ MOCS แต่อย่างไรก็ตามเมื่อทำการทดสอบโดยข้อมูลจำนวน 400 งาน พบว่า การจัดตารางงานโดยใช้ MOCS ให้ค่าเวลารวมที่น้อยที่สุดเมื่อเปรียบเทียบกับวิธีการอื่น คือ MOCS ให้ค่าเวลารวมที่น้อยกว่า 3.69%, 3.77%, 8.82%, 13.85%, 20.77% 23.41% และ 82.39% เมื่อเปรียบเทียบกับวิธีการจัดตารางงานโดยใช้ FCFS, MOABCQ_LJF, MOABCQ_FCFS, Q-learning, HABC_LJF, MOPSO และ Max-Min ตามลำดับ

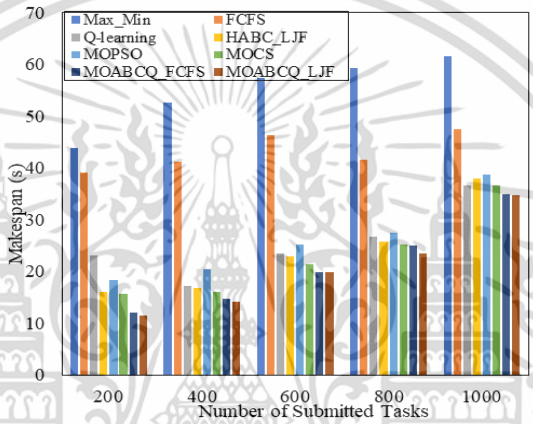
หากพิจารณาข้อมูลเชิงลึกในส่วนของ MOABCQ เมื่อทำการเปรียบเทียบระหว่าง MOABCQ_FCFS และ MOABCQ_LJF พบว่า MOABCQ_LJF ให้ค่าเวลาทำงานรวมที่น้อยกว่า MOABCQ_FCFS ในกรณีที่มีการใช้ข้อมูลในการทดลอง 200, 800 และ 1000 งาน แต่ในทางตรงกันข้ามพบกว่าเมื่อใช้ข้อมูลในการทดลองจำนวน 600 งาน พบว่า MOABCQ_FCFS ให้ค่าเวลาทำงานรวมที่น้อยกว่า MOABCQ_LJF เท่ากับ 0.51%



(ก) ชุดข้อมูลสุ่มแบบอิสระ



(ข) ชุดข้อมูล GoCJ



(ค) ชุดข้อมูล Synthetic workload

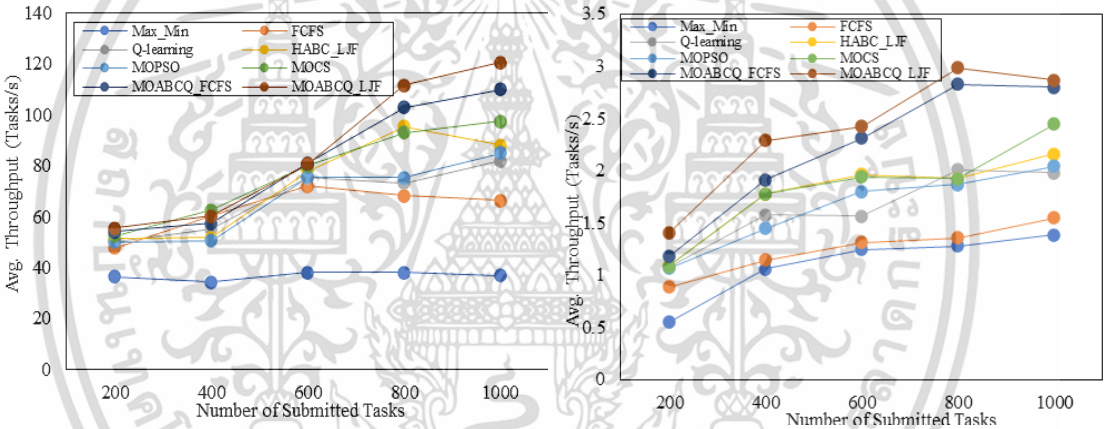
รูปที่ 4.9 เปรียบเทียบประสิทธิภาพของ MOABCQ ในแง่ของเวลาทำงานรวม

เมื่อพิจารณาผลการทดลองในรูปที่ 4.9(ข) และรูปที่ 4.9(ค) แสดงเวลาทำงานรวมเมื่อทำการทดลองโดยใช้ชุดข้อมูล GoCJ และชุดข้อมูล Synthetic workload ตามลำดับ ผลจากการทดลองแสดงให้เห็นว่า การจัดตารางงานโดยใช้ MOABCQ_LJF ให้ค่าเวลาทำงานรวมที่น้อยที่สุด ซึ่งเมื่อทำการทดสอบโดยใช้ชุดข้อมูล GoCJ วิธีการจัดตารางงานโดยใช้ MOABCQ_LJF ให้ค่าเวลาทำงานรวมน้อยกว่าโดยประมาณ 117.80%, 92.15%, 46.17%, 42.25%, 34.94%, 30.62%, and 8.21% เมื่อเปรียบเทียบกับวิธีการจัดตารางงานโดยใช้ Max-Min, FCFS, Q-learning, MOPSO, HABC_LJF, MOCS และ MOABCQ_FCFS ตามลำดับ และเมื่อทำการทดสอบประสิทธิภาพของวิธีการที่นำเสนอโดยใช้ชุดข้อมูล Synthetic workload พบว่าวิธีการจัดตารางงานโดยใช้ MOABCQ_LJF ให้ค่าเวลาทำงานรวมน้อยกว่าโดยประมาณ 150.75%, 98.61%, 25.53%, 22.87%, 15.35%, 10.76%, and 2.97% เมื่อเปรียบเทียบกับ Max-Min, FCFS, MOPSO, Q-learning, HABC_LJF, MOCS, and MOABCQ_FCFS ตามลำดับ ซึ่งสามารถระบุได้ว่า MOABCQ_LJF มีประสิทธิภาพเหนือกว่าวิธีการอื่นที่นำมาเปรียบเทียบทั้ง 2 ชุดข้อมูลที่นำมาทดสอบ ซึ่งผลที่ได้แตกต่างกันที่อัตราการลดลงของเวลาทำงานรวม

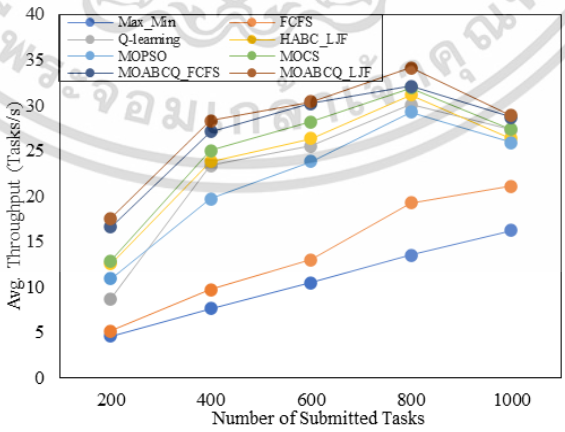
ทั้งนี้เมื่อตรวจสอบในภาพรวมแสดงให้เห็นว่าวิธีการที่นำเสนอนั้นให้ค่าเวลาทำงานรวมที่มีค่าน้อยที่สุด เนื่องจาก MOABCQ_LJF สามารถจัดสรรงานให้กับทรัพยากรที่เหมาะสม จากผลลัพธ์ทั้งหมดเหล่านี้ สามารถสรุปได้ว่าวิธีการจัดตารางงานที่นำเสนอนั้นมีศักยภาพในการจัดสรรทรัพยากรอย่างมีประสิทธิภาพ เพื่อให้ระบบสามารถใช้งานทรัพยากรที่มีอยู่อย่างคุ้มค่ามากที่สุด

4.6.4.2 ผลการทดสอบประสิทธิภาพในกรณีปริมาณงานต่อหน่วยเวลา

ผลการเปรียบเทียบประสิทธิภาพของวิธีการที่นำเสนอในแง่ของปริมาณงานต่อหน่วยเวลา ซึ่งในการทดลองได้กำหนดจำนวนเครื่องคอมพิวเตอร์เสมือนเท่ากับ 100 เครื่อง และกำหนดปริมาณงานในการทดลองมีค่าเท่ากับ 200, 400, 600, 800 และ 1000 งาน ในการทดลองได้มีการทดสอบกับชุดข้อมูลจำนวน 3 ชุดที่ได้กล่าวถึงในหัวข้อที่ 4.6.3 ผลการทดลองแสดงดังรูปที่ 4.10



(ก) ชุดข้อมูลสุ่มแบบอิสระ (ข) ชุดข้อมูล GoCJ



(ค) ชุดข้อมูล Synthetic workload

รูปที่ 4.10 เปรียบเทียบประสิทธิภาพของ MOABCQ ในแง่ของปริมาณงานต่อหน่วยเวลา

ผลจากการทดสอบประสิทธิภาพในมุมของปริมาณต่อหน่วยเวลา โดยใช้ชุดข้อมูลสุ่มแบบอิสระ แสดงผลการทดลองได้ดังรูปที่ 4.10(ก) พบว่าวิธีการจัดตารางงานโดยใช้ MOABCQ สามารถให้ค่าปริมาณงานต่อหน่วยเวลาดีกว่าขั้นตอนวิธีการอื่นที่นำมาเปรียบเทียบ ยกเว้นแต่ในกรณีที่ทำการทดสอบกับปริมาณงานจำนวน 400 งาน พบว่าการจัดตารางงานโดยใช้วิธี MOCS ให้ค่าปริมาณงานต่อหน่วยเวลาที่ดีกว่าวิธีการอื่น กล่าวคือ MOCS มีค่ามากกว่าวิธีการอื่นที่นำมาเปรียบเทียบซึ่งมีค่าเท่ากับ 3.56%, 3.63%, 8.10%, 12.16%, 17.20%, 18.97%, and 45.17% เมื่อเปรียบเทียบกับวิธีการจัดตารางงานโดยใช้ FCFS, MOABCQ_LJF, MOABCQ_FCFS, Q-learning, HABC_LJF, MOPSO และ Max-Min ตามลำดับ และเมื่อพิจารณาถึงวิธีการจัดตารางโดยใช้ MOABCQ ในเชิงลึก พบว่า MOABCQ_LJF ให้ค่าปริมาณงานต่อหน่วยเวลาที่มากกว่า MOABCQ_FCFS ในกรณีเมื่อทำการทดสอบกับงานจำนวน 200, 800 และ 1000 งาน ยกเว้นแต่กรณีที่ทำการทดสอบด้วยงานจำนวน 600 งาน พบว่า MOABCQ_FCFS ให้ค่าปริมาณงานต่อหน่วยเวลาที่มากกว่า MOABC_LJF มีค่าเท่ากับ 0.50%

หลังจากนั้นเมื่อทำการทดสอบประสิทธิภาพของวิธีการที่นำเสนอโดยชุดข้อมูล GoCJ และ Synthetic workload เพื่อทดสอบประสิทธิภาพในมุมของปริมาณงานต่อหน่วยเวลา ผลจากการทดลองแสดงได้ดังรูปที่ 4.10(ข) และ 4.10(ค) ตามลำดับ ซึ่งจากการทดลองพบว่าการจัดตารางงานโดยใช้วิธี MOABCQ มีประสิทธิในการทำงานที่ดีกว่าวิธีการอื่นที่นำมาเปรียบเทียบ และเมื่อพิจารณาประสิทธิภาพในการจัดตารางงานโดยใช้ MOABCQ_FCFS และ MOABCQ_LJF พบว่า MOABCQ_LJF มีประสิทธิภาพในการทำงานที่เหนือกว่า MOABCQ_FCFS ประมาณ 6.14%

ซึ่งเมื่อเปรียบเทียบผลการทดลองในมุมของค่าปริมาณงานต่อหน่วยเวลา เมื่อทำการทดสอบโดยใช้ชุดข้อมูลที่หลากหลาย สามารถสรุปได้ว่า การจัดตารางงานโดยใช้ MOABCQ_LJF มีศักยภาพในการจัดสรรงานให้กับทรัพยากรที่มีอยู่อย่างเหมาะสมเช่นเดียวกับผลการทดลองที่กล่าวมาในข้างต้น

4.6.4.3 ผลการทดสอบประสิทธิภาพในกรณีอัตราส่วนการใช้ทรัพยากรโดยเฉลี่ย

ผลการเปรียบเทียบประสิทธิภาพของวิธีการที่นำเสนอในแง่ของอัตราส่วนการใช้ทรัพยากรโดยเฉลี่ย ซึ่งในการทดลองได้กำหนดจำนวนเครื่องคอมพิวเตอร์เสมือนเท่ากับ 100 เครื่อง และกำหนดปริมาณงานในการทดลองมีค่าเท่ากับ 1000 งาน ในการทดลองได้มีการทดสอบกับชุดข้อมูลจำนวน 3 ชุดที่ได้กล่าวถึงในหัวข้อที่ 4.6.3 ซึ่งผลการทดลองแสดงได้ดังตารางที่ 4.23

ตารางที่ 4.23 เปรียบเทียบประสิทธิภาพของ MOABCQ ในแง่อัตราส่วนการใช้ทรัพยากรโดยเฉลี่ย

Dataset	Task Scheduling Approach							
	Max-Min	FCFS	Q-learning	HABC_LJF	MOPSO	MOCS	MOABCQ_FCFS	MOABCQ_LJF
Random	0.200	0.332	0.660	0.642	0.637	0.698	0.762	**0.812**
GoCJ	0.190	0.252	0.610	0.692	0.684	0.702	0.742	**0.792**
Synthetic workload	0.178	0.505	0.755	0.723	0.722	0.761	0.796	**0.801**

จากผลการทดลองในตารางที่ 4.23 แสดงให้เห็นว่า เมื่อทำการทดสอบโดยใช้ชุดข้อมูลทั้ง 3 ชุด การจัดตารางงานเพื่อเข้าทำงานในเครื่องคอมพิวเตอร์เสมือนโดยใช้วิธี MOABCQ ให้ค่าของอัตราส่วนการใช้ทรัพยากรโดยเฉลี่ยที่มากกว่าการจัดตารางเข้าทำงานโดยใช้วิธีการอื่นที่นำมาเปรียบเทียบ และหากพิจารณาในเชิงลึกทั้ง 3 ชุดข้อมูล พบว่าอัลกอริทึม Q-learning, HABC_LJF, MOPSO, MOCS, MOABCQ_FCFS และ MOABCQ_LJF นั้น ให้ค่าของอัตราส่วนการใช้ทรัพยากรโดยเฉลี่ยที่ใกล้เคียงกัน ซึ่งแตกต่างจากวิธีการจัดตารางงานโดยใช้ Max-Min และ FCFS ผลการทดลองเมื่อทำการทดสอบโดยใช้ชุดข้อมูลสุ่มแบบอิสระ พบว่าการจัดตารางงานโดยใช้วิธี MOABCQ_LJF มีค่าอัตราส่วนการใช้ทรัพยากรโดยเฉลี่ยที่มากกว่า 6.15%, 14.01%, 18.72%, 20.94% และ 21.53% เมื่อเปรียบเทียบกับ MOABCQ_FCFS, MOCS, Q-learning, HABC_LJF และ MOPSO ตามลำดับ

ผลการทดลองในแง่มุมของอัตราส่วนการใช้ทรัพยากรโดยเฉลี่ย เมื่อทำการทดสอบโดยใช้ชุดข้อมูล GoCJ พบว่า การจัดตารางงานโดยใช้วิธี MOABCQ_LJF มีค่าอัตราส่วนการใช้ทรัพยากรโดยเฉลี่ยที่มากกว่า 6.31%, 11.34%, 12.63%, 13.61% และ 22.99% เมื่อเปรียบเทียบกับ MOABCQ_FCFS, MOCS, HABC_LJF, MOPSO และ Q-learning

ผลการทดลองเมื่อใช้ชุดข้อมูล Synthetic workload พบว่าการจัดตารางงานโดยใช้วิธี MOABCQ_LJF มีค่าอัตราส่วนการใช้ทรัพยากรโดยเฉลี่ยที่มากกว่าการจัดตารางงานโดยใช้ MOABCQ_FCFS, MOCS, Q-learning, HABC_LJF และ MOPSO คิดเป็น 0.62%, 4.99%, 5.74%, 9.74% และ 9.86% ตามลำดับ

ซึ่งเมื่อเปรียบเทียบผลการทดลองในมุมของอัตราส่วนการใช้ทรัพยากรโดยเฉลี่ย เมื่อทำการทดสอบโดยใช้ชุดข้อมูลที่หลากหลาย สามารถสรุปได้ว่า การจัดตารางงานโดยใช้ MOABCQ_LJF สามารถช่วยในการจัดตารางการเข้าทำงานในระบบอย่างมีประสิทธิภาพ และสามารถช่วยในการกระจายงานไปยังทรัพยากรที่มีอยู่ได้อย่างทั่วถึง ซึ่งช่วยให้ระบบมีความสมดุลในการทำงาน

4.6.4.4 ผลการทดสอบประสิทธิภาพในกรณีค่าความไม่สมดุลของการกระจาย

ผลการเปรียบเทียบประสิทธิภาพของวิธีการที่นำเสนอในแง่ของค่าความไม่สมดุลของการกระจาย ซึ่งในการทดลองได้กำหนดจำนวนเครื่องคอมพิวเตอร์เสมือนเท่ากับ 100 เครื่อง และกำหนดปริมาณงานในการทดลองมีค่าเท่ากับ 200, 400, 600, 800 และ 1000 งาน ในการทดลองได้มีการทดสอบกับชุดข้อมูลจำนวน 3 ชุดที่ได้กล่าวถึงในหัวข้อที่ 4.6.3 ซึ่งผลการทดลองแสดงได้ดังตารางที่ 4.22

ตารางที่ 4.22 เปรียบเทียบประสิทธิภาพ MOABCQ ในแง่ของค่าความไม่สมดุลของการกระจาย

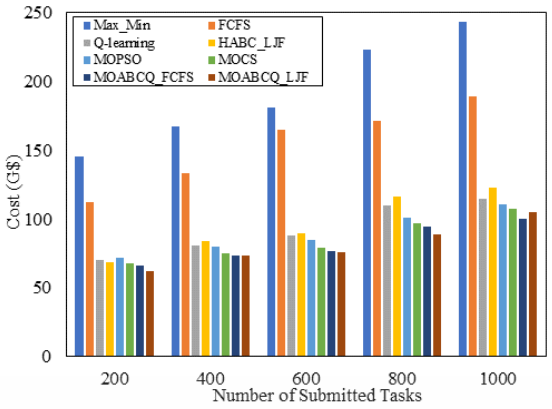
Dataset	Task	Task Scheduling Approach							
		Max_Min	FCFS	Q-learning	HABC_LJF	MOPSO	MOCS	MOABCQ_FCFS	MOABCQ_LJF
Random	200	1.285	0.577	0.442	0.462	0.334	0.280	0.264	**0.200**
	400	0.902	0.425	0.285	0.241	0.309	0.177	0.177	**0.166**
	600	0.916	0.685	0.585	0.378	0.555	0.203	0.134	**0.116**
	800	1.138	0.660	0.237	0.157	0.573	0.177	0.137	**0.115**
	1000	1.330	0.526	0.297	0.175	0.221	0.159	0.097	**0.093**
GoCJ	200	1.023	0.829	0.311	0.430	0.450	0.250	0.270	**0.132**
	400	0.997	0.944	0.272	0.283	0.278	0.204	0.146	**0.123**
	600	1.114	0.652	0.299	0.291	0.299	0.281	0.276	**0.176**
	800	1.234	0.544	0.478	0.294	0.499	0.287	0.387	**0.275**
	1000	1.112	0.912	0.355	0.373	0.387	0.363	0.314	**0.251**
Synthetic workload	200	1.529	0.964	0.225	0.410	0.412	0.378	0.268	**0.187**
	400	1.314	0.905	0.342	0.463	0.472	0.315	**0.293**	0.305
	600	1.130	0.766	0.302	0.734	0.742	0.301	0.418	**0.214**
	800	1.147	0.670	0.295	0.368	0.413	0.274	0.141	**0.117**
	1000	0.834	0.679	0.211	0.235	0.337	0.210	0.209	**0.122**

จากผลการทดลองในตารางที่ 4.22 เมื่อทำการทดสอบโดยใช้ชุดข้อมูลสุ่มแบบอิสระและชุดข้อมูล GoCJ พบว่าการจัดตารางงานโดยใช้วิธีการ MOABCQ_LJF นั้นให้ค่าความไม่สมดุลของการกระจายที่น้อยที่สุดเมื่อเปรียบเทียบกับการจัดตารางงานโดยใช้วิธีการอื่นที่นำมาเปรียบเทียบ ซึ่งบ่งชี้ได้ว่าการจัดตารางงานโดยใช้วิธี MOABCQ_LJF นั้นสามารถกระจายงานไปยังทรัพยากรที่อยู่ในระบบได้อย่างเท่ากันๆ มากกว่าวิธีการอื่นที่นำมาเปรียบเทียบ แต่เมื่อทำการทดลองโดยใช้ชุดข้อมูล Synthetic workload พบว่า เมื่อทำการทดลองโดยกำหนดปริมาณงานเท่ากับ 200, 600, 800 และ 1000 งาน การจัดตารางงานโดยใช้ MOABCQ_LJF นั้นให้ค่าความไม่สมดุลของการกระจายที่น้อยกว่าวิธีการอื่น ยกเว้นแต่กรณีที่กำหนดปริมาณงานในการทดลองมีค่าเท่ากับ 400 งาน กับพบว่า การจัดตารางงานโดยใช้ MOABCQ_FCFS นั้นให้ค่าความไม่สมดุลของการกระจายที่น้อยที่สุด ซึ่งเมื่อทำการเปรียบเทียบกับวิธีการจัดตารางงานโดยใช้ MOABCQ_LJF พบว่า MOABCQ_FCFS สามารถกระจายงานได้ดีกว่า MOABCQ_LJF อยู่ที่ 4.37%

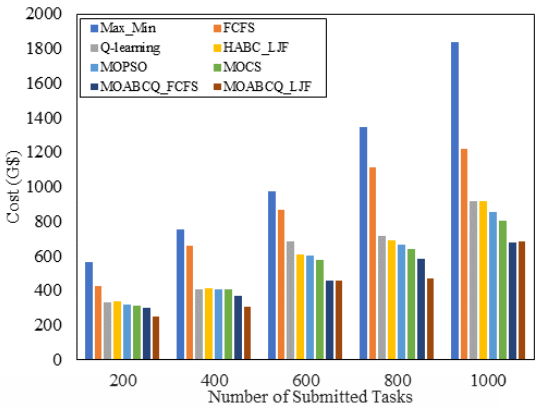
ซึ่งจากผลการทดลองทั้ง 3 ชุดข้อมูลพบว่า การจัดตารางงานโดยใช้ MOABCQ นั้นสามารถช่วยในการกระจายงานไปยังทรัพยากรที่มีอยู่ในระบบได้อย่างทั่วถึง ส่งผลให้ค่าความไม่สมดุลของการกระจายนั้นมีค่าน้อย และหากเมื่อพิจารณาวิธีการที่นำเสนอในเชิงลึกจะพบว่า การจัดตารางงานโดยใช้ MOABCQ_LJF มีประสิทธิภาพในการทำงานที่มากกว่าวิธีการอื่นที่นำมาเปรียบเทียบ แต่ทั้งนี้อย่างไรก็ตามขึ้นอยู่กับชุดข้อมูลที่นำมาทำการทดสอบด้วย

4.6.4.5 ผลการทดสอบประสิทธิภาพในกรณีค่าใช้จ่าย

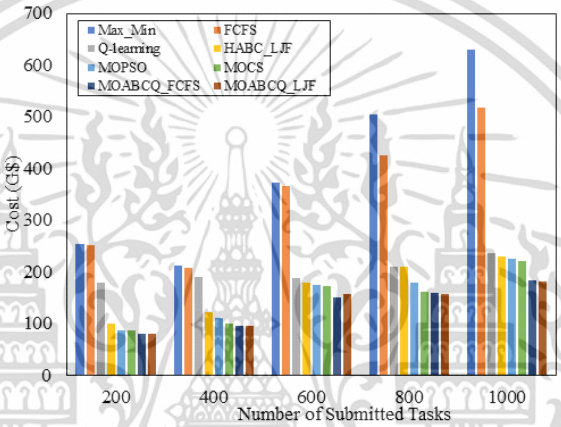
ผลการเปรียบเทียบประสิทธิภาพของวิธีการที่นำเสนอในแง่ของค่าใช้จ่าย ซึ่งในการทดลองได้กำหนดจำนวนเครื่องคอมพิวเตอร์เสมือนเท่ากับ 100 เครื่อง และกำหนดปริมาณงานในการทดลองมีค่าเท่ากับ 200, 400, 600, 800 และ 1000 งาน ในการทดลองได้มีการทดสอบกับชุดข้อมูลจำนวน 3 ชุดที่ได้กล่าวถึงในหัวข้อที่ 4.6.3 ซึ่งผลการทดลองแสดงได้ดังรูปที่ 4.11



(ก) ชุดข้อมูลสุ่มแบบอิสระ



(ข) ชุดข้อมูล GoCJ



(ค) ชุดข้อมูล Synthetic workload

รูปที่ 4.11 เปรียบเทียบประสิทธิภาพของ MOABCQ ในแง่ของค่าใช้จ่าย

เมื่อทำการทดสอบประสิทธิภาพของวิธีที่นำเสนอโดยใช้ชุดข้อมูลแบบสุ่มอิสระ ผลการทดลองประสิทธิภาพแสดงได้ดังรูปที่ 4.11(ก) พบว่า การจัดตารางงานโดยใช้วิธี MOABCQ นั้นสามารถช่วยลดค่าใช้จ่ายที่เกิดขึ้นได้มากกว่าการจัดตารางงานโดยใช้วิธี MOCS, MOPSO, Q-learning, HABC_LJF, FCFS และ Max-Min และเมื่อทำการเปรียบเทียบประสิทธิภาพการทำงานระหว่าง MOABCQ_FCFS และ MOABCQ_LJF พบว่า การจัดตารางงานโดยใช้วิธี MOABCQ_LJF ช่วยลดค่าใช้จ่ายได้มากกว่าการจัดตารางงานโดยใช้วิธี MOABCQ_FCFS ประมาณ 3.38% เมื่อมีการกำหนดปริมาณงานเท่ากับ 200, 400, 600 และ 800 งาน ในทางกลับกันเมื่อกำหนดปริมาณงานเท่ากับ 1000 งาน พบว่า การจัดตารางงานโดยใช้ MOABCQ_LJF นั้นมีค่าใช้จ่ายที่เกิดขึ้นมากกว่า MOABCQ_FCFS ประมาณ 4.88%

เมื่อทำการทดสอบประสิทธิภาพการทำงานโดยใช้ชุดข้อมูล GoCJ ผลการทดลองแสดงได้ดังรูปที่ 4.11(ข) พบว่า การจัดตารางงานโดยใช้วิธี MOABCQ นั้นสามารถช่วยลดค่าใช้จ่ายที่เกิดขึ้นได้มากกว่าการจัดตารางงานโดยใช้วิธีอื่นที่นำมาเปรียบเทียบ เช่นเดียวกับการทดลองโดยใช้ชุด

ข้อมูลสุ่มแบบอิสระ และเมื่อทำการเปรียบเทียบประสิทธิภาพการทำงานระหว่าง MOABCQ_FCFS และ MOABCQ_LJF พบว่า การจัดตารางงานโดยใช้วิธี MOABCQ_LJF ช่วยลดค่าใช้จ่ายในการเข้าใช้บริการได้มากกว่าการจัดตารางงานโดยใช้วิธี MOABCQ_FCFS ประมาณ 20.79% เมื่อมีการกำหนดปริมาณเท่ากับ 200, 400 และ 800 งาน ในทางกลับกันเมื่อกำหนดปริมาณงานเท่ากับ 600 และ 1000 งาน พบว่า การจัดตารางงานโดยใช้ MOABCQ_LJF นั้นมีค่าใช้จ่ายที่เกิดขึ้นมากกว่าการจัดตารางงานโดยใช้วิธี MOABCQ_FCFS ประมาณ 0.48%

เมื่อทำการทดสอบประสิทธิภาพการทำงานโดยใช้ชุดข้อมูล Synthetic workload ผลการทดลองแสดงได้ดังรูปที่ 4.11(ค) พบว่า การจัดตารางงานโดยใช้วิธี MOABCQ ช่วยลดค่าใช้จ่ายที่เกิดขึ้นได้ดีที่สุด เช่นเดียวกับชุดข้อมูลสุ่มแบบอิสระและ GoCJ และเมื่อทำการเปรียบเทียบประสิทธิภาพการทำงานระหว่าง MOABCQ_FCFS และ MOABCQ_LJF นั้นพบว่า การจัดตารางงานโดยใช้วิธี MOABCQ_LJF ช่วยลดค่าใช้จ่ายได้มากกว่าจัดตารางงานโดยใช้วิธี MOABCQ_FCFS ยกเว้นเมื่อกำหนดปริมาณงานเท่ากับ 400 งาน พบว่า การจัดตารางงานโดยใช้ MOABCQ_LJF นั้นมีค่าใช้จ่ายที่เกิดขึ้นมากกว่า MOABCQ_FCFS ประมาณ 1.90%

ซึ่งจากผลการทดลองทั้ง 3 ชุดข้อมูลพบว่า การจัดตารางงานโดยใช้ MOABCQ นั้นสามารถจัดตารางการทำงานได้เหมาะสมเพื่อให้เกิดค่าใช้จ่ายที่ลดลง เมื่อเปรียบเทียบกับวิธีการอื่น และหากเมื่อพิจารณาวิธีการที่นำเสนอในเชิงลึกจะพบว่า MOABCQ_LJF นั้นมีประสิทธิภาพในการจัดตารางงานที่เหมาะสมกับทรัพยากรที่อยู่ในระบบได้มากกว่าวิธีการ MOABCQ_FCFS เมื่อพิจารณาจากค่าอัตราส่วนความแตกต่างระหว่างทั้ง 2 วิธีการ แต่ทั้งนี้ขึ้นอยู่กับชุดข้อมูลที่นำมาใช้ในการทดสอบด้วย

4.7 ค่าความซับซ้อนของวิธี MOABCQ

ค่าความซับซ้อน (Time Complexity) ของวิธี MOABCQ คำนวณได้ดังนี้ ใน ABC มีการกำหนดจำนวนประชากรเริ่มต้น (Initial population) จำนวน n และการแบ่งผึ้งออกเป็น Employed bees และ Onlooker bee ดังนั้น จำนวนรอบในการวนซ้ำแต่ละครั้งเพื่อค้นหาเครื่องคอมพิวเตอร์เสมือน (VMs) ที่เหมาะสมในการประมวลผลแบบกลุ่มเมฆเท่ากับ n และจำนวนในการปรับปรุงข้อมูลในตาราง Q-table มีค่าเท่ากับ n ครั้งเช่นเดียวกัน ส่งผลให้ค่าความซับซ้อนของวิธีการจัดตารางงานโดยใช้ MOABCQ คือ O(n) หาก ABC ทำซ้ำในขั้นตอนนี้จำนวน k ครั้ง ค่าความซับซ้อนจะมีค่าเท่ากับ k x O(n) เนื่องจาก k เป็นค่าคงที่ ดังนั้นค่าความซับซ้อนของวิธี MOABCQ มีค่าเท่ากับ O(n)

บทที่ 5
บทสรุปและข้อเสนอแนะ

งานวิจัยนี้นำเสนอวิธีการจัดตารางงานที่เป็นอิสระต่อกันในประมวลผลแบบกลุ่มเมฆ โดยมุ่งเน้นที่วิธีการจัดตารางงานแบบที่มีหลายวัตถุประสงค์ อีกทั้งยังได้มีการนำวิธีการทางปัญญาประดิษฐ์เข้ามาช่วยในการแก้ไขปัญหา ซึ่งเป็นการประยุกต์ใช้ขั้นตอนอาณานิคมผึ้งเทียม (Artificial Bee Colony algorithm) ควบคู่กับการใช้ขั้นตอนวิธีการเรียนรู้แบบคิว (Q-learning algorithm) วิธีการที่เสนอมีวัตถุประสงค์เพื่อเพิ่มประสิทธิภาพในการจัดตารางงานและการใช้ทรัพยากร และสร้างสมดุลระหว่างเครื่องคอมพิวเตอร์เสมือน โดยจากผลการประเมินประสิทธิภาพสามารถสรุปผลและวิเคราะห์ผลการทดลอง รวมถึงข้อจำกัดและข้อเสนอแนะ ได้ดังนี้

5.1 สรุปผลและวิเคราะห์ผลการทดลอง

งานวิจัยนี้มีวัตถุประสงค์เพื่อเพิ่มประสิทธิภาพการใช้งานทรัพยากรในรูปแบบการประมวลผลแบบกลุ่มเมฆ และช่วยให้มีการกระจายงานระหว่างเครื่องคอมพิวเตอร์เสมือนแบบสมดุล ซึ่งทางผู้วิจัยได้เสนอวิธีการจัดตารางงานโดยการประยุกต์ใช้ขั้นตอนอาณานิคมผึ้งเทียม (Artificial Bee Colony algorithm) ควบคู่กับการจัดตารางงานแบบฮิวริสติก ได้แก่ การจัดเรียงข้อมูลตามลำดับการมาถึงของงาน (FCFS) การจัดเรียงข้อมูลตามขนาดของงานโดยเรียงข้อมูลจากเล็กที่สุดไปหาขนาดใหญ่ที่สุด (LJF) และการจัดเรียงข้อมูลตามขนาดของงานโดยเรียงข้อมูลจากใหญ่ที่สุดไปหาขนาดเล็กที่สุด (SJF) เพื่อช่วยในการจัดตารางงาน ในช่วงแรกของการทดสอบประสิทธิภาพของวิธีการจัดตารางงานที่นำเสนอ ได้มีการพิจารณาจากเวลาที่ใช้ในการทำงานและค่าความสมดุลในระบบการประมวลผลแบบกลุ่มเมฆ ซึ่งเรียกวิธีการที่นำเสนอนี้ว่า “HABC” ในการทดสอบประสิทธิภาพมีการใช้ชุดข้อมูลที่มีขนาดของงานและประเภทของชุดข้อมูลที่มีรูปแบบการกระจายงานที่แตกต่างกัน คือ ชุดข้อมูลแบบสุ่มอิสระ ชุดข้อมูลแบบการกระจายข้อมูลแบบแจกแจงปกติ ชุดข้อมูลแบบการกระจายข้อมูลแบบเบ้ขวา และชุดข้อมูลแบบการกระจายข้อมูลแบบเบ้ซ้าย ซึ่งในการทดสอบประสิทธิภาพได้มีการกำหนดจำนวนของเครื่องคอมพิวเตอร์เสมือนให้มีค่าคงที่ ในขณะที่จำนวนงานในระบบมีการเปลี่ยนแปลง ซึ่งผลจากการทดสอบประสิทธิภาพของระบบพบว่า วิธีการ HABC_LJF นั้นสามารถช่วยลดเวลาในการเข้าใช้งานระบบการประมวลผลแบบกลุ่มเมฆได้อย่างมีประสิทธิภาพ และสามารถทำงานได้ดีกว่าวิธีจัดตารางงานแบบฮิวริสติก อีกทั้งวิธีการที่นำเสนอนั้นมีประสิทธิภาพในการทำงานที่ดีกว่าการจัดตารางงานโดยใช้การหาค่าความเหมาะสมที่สุดด้วยวิธีอาณานิคมมดและขั้นตอนวิธีการหาคำตอบเหมาะสมที่สุดแบบกลุ่มอนุภาค

จากนั้นทางผู้วิจัยได้เสนอวิธีการจัดตารางงานแบบที่มีหลายวัตถุประสงค์ในการประมวลผลแบบกลุ่มเมฆ โดยพิจารณาจากเวลารวมทั้งหมดในการทำงาน ค่าใช้จ่ายที่ใช้ในการเข้าทำงานในเครื่องคอมพิวเตอร์เสมือน และการใช้งานทรัพยากรพร้อมกันน้อยที่สุด ซึ่งได้มีการนำขั้นตอนอาณานิคมผึ้งเทียมควบคู่กับการปรับใช้ขั้นตอนวิธีการเรียนรู้แบบคิว เพื่อช่วยให้การทำงานในขั้นตอนอาณานิคมผึ้งเทียมสามารถทำงานได้เร็วขึ้น เรียกอัลกอริทึมนี้ว่า “MOABCQ” อีกทั้งมีการเพิ่มเติมการจัดตารางงานแบบฮิวริสติก (จัดเรียงข้อมูลตามลำดับการมาถึงของงาน (FCFS) และเรียงข้อมูลจากใหญ่ที่สุดไปหาขนาดเล็กที่สุด (LJF) อีกด้วย ในการประเมินประสิทธิภาพครั้งนี้ได้ทำการทดลองกับชุดข้อมูลที่มีความหลากหลายเพื่อประเมินประสิทธิภาพของวิธีการที่นำเสนอ ได้แก่ ชุดข้อมูลสุ่มแบบอิสระ ชุดข้อมูลของ Google Cloud Jobs (GoCJ) และชุดข้อมูลของ Synthetic workload อีกทั้งยังได้ทำการเปรียบเทียบประสิทธิภาพการจัดตารางงานโดยใช้วิธีการทางฮิวริสติกที่เป็นที่รู้จัก เช่น ขั้นตอนการจัดตารางงานโดยใช้ Max-Min เป็นต้น และยิ่งไปกว่านั้น ยังได้ทำการเปรียบเทียบการจัดตารางงานโดยใช้วิธีการทางเมตาฮิวริสติกที่เป็นที่นิยม คือ ขั้นตอนวิธีการหาคำตอบเหมาะสมที่สุดแบบกลุ่มอนุภาค (PSO) ขั้นตอนวิธีการหาค่าเหมาะสมที่สุดแบบนกกาเหว่า (CS) อีกทั้งยังได้ทำการเปรียบเทียบกับวิธีที่นำเสนอในการทดลองก่อนหน้า คือ วิธีการจัดตารางงานโดยใช้ขั้นตอนอาณานิคมผึ้งเทียมควบคู่กับการจัดลำดับงานแบบฮิวริสติก (HABC_LJF) ผลจากการทดลองพบว่า การจัดตารางโดยใช้วิธี MOABCQ จะสามารถช่วยในการกระจายงานให้กับเครื่องคอมพิวเตอร์เสมือนที่มีอยู่ในระบบได้อย่างเหมาะสม และสามารถช่วยลดเวลาในการทำงานและค่าใช้จ่ายในการใช้งานระบบ เมื่อทำการเปรียบเทียบกับการจัดตารางงานโดยใช้ขั้นตอนวิธีการหาคำตอบโดยใช้ค่าสูงสุดต่ำสุด (Max-Min) วิธีการจัดตารางงานแบบลำดับการมาถึงของงาน (FCFS) ขั้นตอนวิธีการเรียนรู้แบบคิว (Q-learning) ขั้นตอนวิธีการหาค่าเหมาะสมโดยใช้ขั้นตอนวิธีอาณานิคมมด (ACO) ขั้นตอนวิธีการหาคำตอบเหมาะสมที่สุดแบบกลุ่มอนุภาค (PSO) และขั้นตอนวิธีการหาค่าเหมาะสมที่สุดแบบนกกาเหว่า (CS) ซึ่งผลจากการทดลองแสดงให้เห็นว่า การจัดตารางโดยใช้วิธี MOABCQ มีประสิทธิภาพการทำงานที่ดีกว่าวิธีการจัดตารางงานที่นำมาเปรียบเทียบ

5.2 ข้อจำกัด

ในการศึกษาครั้งนี้มีวัตถุประสงค์เพื่อเพิ่มประสิทธิภาพการใช้งานของทรัพยากรของระบบการประมวลผลแบบกลุ่มเมฆ และให้มีการกระจายงานระหว่างเครื่องคอมพิวเตอร์เสมือนแบบสมดุล อีกทั้งยังลดเวลารวมในการทำงานของระบบและลดค่าใช้จ่ายที่เกิดขึ้นโดยการนำวิธีการทางปัญญาประดิษฐ์เข้ามาช่วยในการแก้ไขปัญหา เพื่อหาวิธีการที่เหมาะสมในการทำงาน และแม้ว่าขั้นตอนวิธีการจัดตารางที่นำเสนอจะมีประสิทธิภาพในการศึกษาเป็นอย่างมาก แต่ก็ยังคงพบข้อจำกัดในการทำงานบางกรณี ดังต่อไปนี้

1. การประเมินประสิทธิภาพของวิธีการที่นำเสนอใช้ชุดข้อมูลในการทดสอบจำนวน 3 ชุด ได้แก่ ชุดข้อมูลสุ่มแบบอิสระ ชุดข้อมูลของ Google Cloud Jobs (GoCJ) และชุดข้อมูล Synthetic

workload ในกรณีที่มีการเปลี่ยนแปลงชุดข้อมูลหรือรูปแบบของข้อมูลที่นำมาทดสอบ อาจส่งผลให้ขั้นตอนวิธีการจัดตารางงานที่นำเสนอนั้น อาจไม่สามารถเพิ่มประสิทธิภาพการเข้าใช้งานทรัพยากรหรือลดค่าใช้จ่ายที่เกิดขึ้นจากการเข้าใช้ระบบการประมวลผลแบบกลุ่มเมฆได้ดีเท่าที่ควร

2. แม้ว่าในการศึกษาครั้งนี้ ทางผู้วิจัยได้ทำการทดลองกับชุดข้อมูลของงานที่มีความหลากหลายและมีขนาดของงานที่แตกต่างกัน แต่ในกรณีที่ชุดข้อมูลนั้นมีขนาดของงานที่มีความใกล้เคียงหรือเกาะกลุ่มกันมาก ส่งผลการจัดตารางงานโดยใช้วิธี MOABCQ นั้นไม่สามารถเพิ่มประสิทธิภาพการเข้าใช้งานทรัพยากร หรือลดเวลาการเข้าทำงานในระบบการประมวลผลแบบกลุ่มเมฆได้ดีเท่าที่ควร

3. การจัดตารางโดยใช้วิธี MOABCQ มีประสิทธิภาพการทำงานที่ดีกว่าวิธีการจัดตารางงานที่นำมาเปรียบเทียบ แต่ทั้งนี้ไม่สามารถยืนยันได้ว่าการจัดตารางงานโดยใช้วิธี MOABCQ_LJF มีประสิทธิภาพในการทำงานมากที่สุด เนื่องจากไม่สามารถเพิ่มประสิทธิภาพการทำงานของระบบได้กับทุกชุดข้อมูลที่นำมาใช้ในการทดสอบ

4. การประเมินประสิทธิภาพของวิธีการจัดตารางที่นำเสนอในครั้งนี้ ได้การเปรียบเทียบความสามารถการทำงานของระบบในแง่มุมที่หลากหลาย ได้แก่ เวลาการทำงานรวมเฉลี่ย การวัดค่าความไม่สมดุลของการกระจายข้อมูล ค่าใช้จ่าย อัตราส่วนการใช้งานทรัพยากรโดยเฉลี่ย ปริมาณงานต่อหน่วยเวลา เป็นต้น ซึ่งทั้งนี้ในการประเมินประสิทธิภาพของวิธีการที่นำเสนอนั้นได้ทำการทดสอบเฉพาะกรณีที่จำนวนงานในระบบมีการเปลี่ยนแปลง โดยกำหนดจำนวนงานในการทดลองมีค่าระหว่าง 100 ถึง 1,500 งาน และกำหนดให้จำนวนเครื่องคอมพิวเตอร์เสมือนมีค่าคงที่ตลอดการทดลอง ซึ่งในการประเมินประสิทธิภาพได้กำหนดจำนวนเครื่องคอมพิวเตอร์เสมือนมีค่าเท่ากับ 60 เครื่อง และจำนวน 100 เครื่อง ซึ่งผลจากการทดลองพบว่าวิธีการที่นำเสนอนั้นมีความเหมาะสมในการจัดตารางเข้าทำงานในการประมวลผลแบบกลุ่มเมฆ ในกรณีที่มีการกำหนดจำนวนงานในช่วงดังกล่าว แต่ไม่สามารถยืนยันได้ว่าหากมีการเปลี่ยนแปลงจำนวนงานมากกว่าที่มีการทดลองหรือมีการเปลี่ยนแปลงจำนวนเครื่องคอมพิวเตอร์เสมือนนั้น วิธีการที่นำเสนอยังสามารถเพิ่มประสิทธิภาพการทำงานของระบบการประมวลผลแบบกลุ่มเมฆได้ดีกว่าขั้นตอนวิธีการอื่นที่นำมาเปรียบเทียบหรือไม่

5. การประเมินประสิทธิภาพของวิธีการที่นำเสนอครั้งนี้ ทำการทดลองภายใต้การสร้างสภาพแวดล้อมจำลองแบบการประมวลผลแบบกลุ่มเมฆแบบส่วนตัว ซึ่งหากมีการเปลี่ยนแปลงสภาพแวดล้อมของการประมวลผลแบบกลุ่มเมฆหรือรูปแบบการมวลผลแบบกลุ่มเมฆแบบต่างๆ วิธีการที่นำเสนออาจไม่สามารถเพิ่มประสิทธิภาพการทำงานของระบบได้ดีเท่าที่ควร

5.3 ข้อเสนอแนะ

จากข้อจำกัดที่กล่าวมาในข้างต้นหากในการศึกษาครั้งต่อไปสามารถนำมาแก้ไขหรือปรับปรุงต่อยอดในการพัฒนา เพื่อลดข้อจำกัดในการศึกษาจะสามารถทำให้ได้วิธีการจัดตารางงานที่เหมาะสม เพื่อให้สามารถเพิ่มประสิทธิภาพการทำงานของระบบการประมวลผลแบบกลุ่มเมฆได้มากขึ้น และนำไปสู่ทิศทางที่น่าสนใจมากขึ้นดังนี้

1. ผู้วิจัยควรทำการทดสอบเพิ่มเติมโดยการเพิ่มปริมาณงานที่ทำการทดสอบ อีกทั้งควรทำการทดสอบในกรณีที่จำนวนเครื่องคอมพิวเตอร์เสมือนมีการเปลี่ยนแปลง เพื่อให้ทราบว่าวิธีการที่นำเสนอยังสามารถเพิ่มประสิทธิภาพการทำงานของระบบการประมวลผลแบบกลุ่มเมฆได้ดีกว่าขั้นตอนวิธีการอื่นที่นำมาเปรียบเทียบหรือไม่ หากวิธีการที่นำเสนอไม่ดีกว่าอาจพยายามปรับปรุงวิธีการจัดตารางเพื่อให้มีความเหมาะสมมากยิ่งขึ้น

2. ควรทดสอบกับชุดข้อมูลที่มีความหลากหลายมากขึ้น หรือพิจารณาข้อมูลที่นำมาทำการทดสอบโดยละเอียด เพื่อหาวิธีการหรือแนวทางในการปรับปรุงขั้นตอนในการทำงาน เพื่อให้ได้ผลลัพธ์ที่มีประสิทธิภาพมากขึ้น

3. ในปัจจุบันนี้รูปแบบหรือประเภทของระบบการประมวลผลแบบกลุ่มเมฆมีความหลากหลายมากขึ้น ดังนั้นการจัดตารางงานในสภาพแวดล้อมแบบคลาวด์จากผู้ให้บริการที่หลากหลาย (Multi-Cloud) หรือ Fog cloud หรือ Edge cloud ถือเป็นสิ่งที่ท้าทายและน่าสนใจในการทำงาน อาจเสนอวิธีการจัดตารางให้เหมาะสมกับสภาพแวดล้อมในแบบต่างๆ อีกทั้งในการทดสอบอาจมีการทดสอบในสภาพแวดล้อมจริง เพื่อดูประสิทธิภาพการทำงานของวิธีการที่นำเสนอด้วย

4. ควรมีการนำวิธีการเรียนรู้ของเครื่องวิธีการอื่นๆ มาปรับใช้เพิ่มเติม เพื่อเพิ่มประสิทธิภาพการเข้าใช้งานในระบบการประมวลผลแบบกลุ่มเมฆ

เอกสารอ้างอิง

[1] George, S. and Kumar, V. P. S. 2012. "Multicloud computing for on-demand resource provisioning using clustering." pp. 1-6. in **Proceedings of IET Chennai 3rd International on Sustainable Energy and Intelligent Systems (SEISCON 2012)**. Tiruchengode : IET.

[2] Yang, S. Pan, L. Wang, Q. Liu, S. and Zhang S. 2018. "Subscription or Pay-as-You-Go: Optimally Purchasing IaaS Instances in Public Clouds." pp. 219-226. in **Proceedings of 2018 IEEE International Conference on Web Services (ICWS)**. San Francisco, CA, USA : IEEE.

[3] Ardagna, D. G. Casale, M. Ciavotta, Perez, J. F. and Wang, W. 2014. "Quality-of-service in cloud computing: modeling techniques and their applications." *J. Internet. Serv. Appl.* 5(11) : 1-17.

[4] Psychas, K. and Ghaderi, J. 2018. "On Non-Preemptive VM Scheduling in the Cloud." *SIGMETRICS Perform. Eval. Rev.* 46(1) : 67-69.

[5] Chen,Y. Ganapathi, A. Griffith, R. and Katz, R. 2010. "**Analysis and Lessons from a Publicly Available Google Cluster Trace.**" RAD Lab, EECS Dep. Univ. California, Berkeley Technical Report UCB/EECS-2010-95 94.

[6] Karaboga, D. 2005. "**An Idea Based on Honey Bee Swarm for Numerical Optimization.**" ERU. Kayseri, Turkey Technical Report-tr06.

[7] Akay, B. and Karaboga, D. 2012. "A modified Artificial Bee Colony algorithm for real-parameter optimization." *Inf. Sci.* 192(1) : 120-142.

[8] Karaboga, D. and Gorkemli, B. 2011. "A combinatorial Artificial Bee Colony algorithm for traveling salesman problem." pp. 50-53. in **Proceedings of 2011 International Symposium on Innovations in Intelligent Systems and Applications**. Istanbul, Turkey : IEEE.

[9] Li, X. Peng, Z. Du, B. Guo, J. Xu, W. and Zhuang, K. 2017. "Hybrid artificial bee colony algorithm with a rescheduling strategy for solving flexible job shop scheduling problems." *Comput. Ind. Eng.* 113 : 10-26.

[10] Sutton, R.S. and Barto, A.G. 2018. **Reinforcement Learning: An Introduction**. 2nd ed. London : MIT Press.

เอกสารอ้างอิง (ต่อ)

[11] Hussain, A. and Aleem, M. 2018. "GoCJ: Google Cloud Jobs Dataset for Distributed and Cloud Computing Infrastructures." *Data.* 3(4) : 38.

[12] Hussain, A. Aleem, M. Khan, A. Iqbal, M. A. and Islam, M. A. 2018. "RALBA: a computation-aware load balancing scheduler for cloud computing." *Cluster Comput.* 21(3) : 1667–1680.

[13] Buyya, R. Yeo, C.S. Venugopal, S. Broberg, J. and Brandic, I. 2009. "Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility." Future. Gener. Comp. Syst. 25(6) : 599–616.

[14] 2021. **Cloud Computing Tutorial.** [Online]. Available : https://www.javatpoint.com/cloud-computing-tutorial.

[15] Wieder, P. Butler, J.M. Theilmann, W. and Yahyapour, R., editors. 2011. **Service Level Agreements for Cloud Computing**. New York : Springer.

[16] Tang, K. Zang, J. M. and Feng, C. H. 2011. "Application Centric Lifecycle Framework in Cloud." pp. 329 – 334. in **Proceedings of IEEE 8th Int. Conf. on e-Business Engineering (ICEBE).**

[17] Givehchi, O. and Jasperneite, J. 2013. "Industrial Automation Services as part of the Cloud: First experiences." in **Proceedings of Jahreskolloquium Kommunikation in der Automation - KommA, Magdeburg.**

[18] Hamilton, E. 2019. **Hybrid Cloud: A 2021 Guide to a New Approach**. [Online]. Available : https://www.cloudwards.net/hybrid-cloud/ .

[19] Dutta K. 2012. "A Smart Job Scheduling System for Cloud Computing Service Providers and Users: Modeling and Simulation." in **Proceedings of 2012 1st International Conference on Recent Advances in Information Technology (RAIT).** Dhanbad, India : IEEE.

[20] Sindhu S. and Mukherjee S. 2011. "Efficient Task Scheduling Algorithms for Cloud Computing Environment." pp. 79-83. in Mantri, A. Nandi, S. Kumar, G. and Kumar, S. (eds) **High Performance Architecture and Grid Computing. HPAGC 2011. Communications in Computer and Information Science**. 169. Berlin, Heidelberg : Springer.

เอกสารอ้างอิง (ต่อ)

[21] Kwesterh. 2010. Hardware Virtualization. [Online]. Available : https://en.wikiversity.org/wiki/File:Hardware_Virtualization.JPG .

[22] Ertel, W. 2017. **Introduction to Artificial Intelligence.** Springer.

[23] Patducjacquet. 2017. "Machine Learning (ML) : a quick overview." [Online]. Available : https://patducjacquet.wordpress.com/2017/06/03/machine-learning-ml-a-quick-overview/.

[24] Phan, D.T. Ghanbarzadeh, A. Koc, E. Otri, S. Rahim, S. and Zaidi, M. 2005. "**Bee Algorithm A Novel Approach to Function Optimisation.**" The Manufacturing Engineering Centre Cardiff University Queen's University Technical Note: MEC 0501.

[25] Phan, D.T. Ghanbarzadeh, A. Koc, E. Otri, S. Rahim, S. and Zaidi, M. 2006. "The Bees Algorithm - A Novel Tool for Complex Optimisation Problems." pp. 454-459. in Pham, D.T. Eldukhri, E.E. and Soroka, A.J. (eds). **Intelligent Production Machines and Systems.** Cardiff, UK : Elsevier.

[26] Williams, C. 2009. "Rethinking the bee's waggle dance." [Online]. Available : https://www.newscientist.com/article/mg20327262-400-rethinking-the-bees-waggle-dance/.

[27] Dorigo, M. Maniezzo, V. and Colorni A. "Ant System: Optimization by a colony of cooperating agents". *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 26(1):29–41, 1996.

[28] Dorigo, M. and Stützle, T. "Ant Colony Optimization." MIT Press, Cambridge, MA, 2004.

[29] Bonyadi, M. R. and Michalewicz, Z. "Particle Swarm Optimization for Single Objective Continuous Space Problems: A Review," in *Evolutionary Computation*, vol. 25, no. 1, pp. 1-54, March 2017, doi: 10.1162/EVCO_r_00180.

[30] Kennedy J. (2011) Particle Swarm Optimization. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-30164-8_630

เอกสารอ้างอิง (ต่อ)

[31] Angeline PJ (1998a) Evolutionary optimization versus particle swarm optimization philosophy and performance difference. In: Evolutionary programming, Lecture notes in computer science, vol. vii edition. Springer, Berlin

[32] Al-kazemi B, Mohan CK (2002) Multi-phase generalization of the particle swarm optimization algorithm. In: Proceedings of 2002 IEEE Congress on Evolutionary Computation, Honolulu, Hawaii, August 7–9, pp 489–494

[33] Rajabioun, R. (2011). Cuckoo Optimization Algorithm. *Applied Soft Computing, 11*(8), 5508-5518.

[34] X.-S. Yang; S. Deb (December 2009). *Cuckoo search via Lévy flights*. World Congress on Nature & Biologically Inspired Computing (NaBIC 2009). IEEE Publications. pp. 210–214. arXiv:1003.1594v1.

[35] Humphries, Nicolas E.; Queiroz, Nuno; Dyer, Jennifer R. M.; Pade, Nicolas G.; Musyl, Michael K.; Schaefer, Kurt M.; Fuller, Daniel W.; Brunnschweiler, Juerg M.; Doyle, Thomas K.; Houghton, Jonathan D. R.; Hays, Graeme C.; Jones, Catherine S.; Noble, Leslie R.; Wearmouth, Victoria J.; Southall, Emily J.; Sims, David W. (2010). *"Environmental context explains Lévy and Brownian movement patterns of marine predators" (PDF). Nature. **465** (7301): 1066–1069. Bibcode:2010Natur.465.1066H.*

[36] Sims, David W.; Reynolds, Andrew M.; Humphries, Nicholas E.; Southall, Emily J.; Wearmouth, Victoria J.; Metcalfe, Brett; Twitchett, Richard J. (July 29, 2014). "Hierarchical random walks in trace fossils and the origin of optimal search behavior". Proceedings of the National Academy of Sciences. 111 (30): 11073–11078. doi:10.1073/pnas.1405966111. ISSN 0027-8424. PMC 4121825. PMID 25024221.

[37] Wikipedia. "Multi-objective optimization." [Online]. Available : https://en.wikipedia.org/wiki/Multi-objective_optimization.

[38] Konak, A. Coit, D.W. and Alice, E.S. 2006. "Multi-objective optimization using genetic algorithms : A tutorial." *Reliability Engineering and System Safety.* 91(9) : 992-1007.

เอกสารอ้างอิง (ต่อ)

[39] Boonlong, K. 2011. "Genetic Algorithms for Multi-objective Optimization." Burapha Sci. J. 16(1) : 107-114.

[40] Calheiros, R. N. Ranjan, R. Rose, C.A.F.D. and Buyya, R. 2009. "**CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services**." arXiv preprint arXiv: 0903.2525.

[41] Calheiros, R. N. Ranjan, R. Rose, C. A. F. D. and Buyya, R. 2010. "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms." *Softw. Pract. Exper.* 41(1) : 23-50.

[42] J. Ullman, "NP-Complete Scheduling Problems," *J. Comput. Syst. Sci.,* vol. 10, pp.384-393, 1975.

[43] Islam, T. and Hasan, M. S. 2017. "A performance comparison of load balancing algorithms for cloud computing," pp. 130-135. in **Proceedings of International Conference on the Frontiers and Advances in Data Science (FADS).** China : IEEE.

[44] Hazra, D. Roy, A. Midya, S. and Majumder, K. 2018. "Distributed Task Scheduling in Cloud Platform: A Survey." pp. 183-191. In Satapathy, S. Bhateja, V. and Das, S. (eds). **Smart Computing and Informatics. Smart Innovation, Systems and Technologies**. Vol 77. Singapore : Springer.

[45] Kalra, M. and Singh, S. 2015. "A review of metaheuristic scheduling techniques in cloud computing." *Egypt. Inform. J.* 16(3) : 275-295.

[46] Zuo, L. Shu, L. Dong, S. Zhu, C. and Hara, T. 2015. "A Multi-Objective Optimization Scheduling Method Based on the Ant Colony Algorithm in Cloud Computing." *IEEE Access*. 3 : 2687-2699.

[47] Guo, X. 2021. "Multi-objective task scheduling optimization in cloud computing based on fuzzy self-defense algorithm." *Alexandria Eng. J.* 60(6) : 5603-5609.

[48] Sanaj, M. and Prathap, P. J. 2020. "An efficient approach to the map-reduce framework and genetic algorithm based whale optimization algorithm for task scheduling in cloud computing environment." *Mater. Today, Proc.* 37 : 3199-3208.

เอกสารอ้างอิง (ต่อ)

[49] Farahnakian, F. Liljeberg, P. and Plosila, J. 2014. "Energy-Efficient Virtual Machines Consolidation in Cloud Data Centers Using Reinforcement Learning." pp. 500-507. in **Proceedings of 2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based.** Italy : IEEE.

[50] Rugwiro, U. Gu, C. and Ding, W. 2019. "Task Scheduling and Resource Allocation Based on Ant-Colony Optimization and Deep Reinforcement Learning." *J. Internet Technol.* 20(5) : 1463-1475.

[51] Fang, Y. Wang, F. and Ge, J. 2010. "A Task Scheduling Algorithm Based on Load Balancing in Cloud Computing." pp. 271-277. In Wang, F.L. Gong, Z. Luo, X. Lei, J. (eds). **Web Information Systems and Mining. WISM 2010. Lecture Notes in Computer Science**. Vol 6318. Berlin : Springer.

[52] Gan, G. Huang, T. and Gao, S. 2010. "Genetic simulated annealing algorithm for task scheduling based on cloud computing environment." pp. 60-63. in **Proceedings of 2010 International Conference on Intelligent Computing and Integrated Systems**.

[53] Patel, G. Mehta, R. and Bhoi, U. R. 2015. "Enhanced Load Balanced Min-min Algorithm for Static Meta Task Scheduling in Cloud Computing." *Procedia Comput. Sci. 57 :* 45-553.

[54] Shrimali, B. and Patel, H. 2020. "Multi-objective optimization oriented policy for performance and energy efficient resource allocation in Cloud environment." *J. King Saud Univ. Comput. Inf. Sci.* 32(7) : 860-869, 2020.

[55] Zhang, H. Shi, J. Deng, B. Jia, G. Han, G. and Shu, L. 2019. "MCTE: Minimizes Task Completion Time and Execution Cost to Optimize Scheduling Performance for Smart Grid Cloud." *IEEE Access*. 7 : 134793-134803.

[56] Li, K. Xu, G. Zhao, G. Dong, and Y. Wang, D. 2011. "Cloud task scheduling based on load balancing ant colony optimization." pp. 3–9. in **Proceedings of IEEE 6th Annual China Grid Conference**. China : IEEE.

[57] Tawfeek, M. A. El-Sisi, A. Keshk, A. and Torkey, F. 2013. "Cloud task scheduling based on ant colony optimization." pp. 64-69. in **Proceedings of 8th Int. Conf. Comput. Eng. Sys. (ICCES)**.

เอกสารอ้างอิง (ต่อ)

[58] Awad, A.I. El-Hefnawy, N.A. and Abdel_kader, H.M. 2015. "Enhanced Particle Swarm Optimization for Task Scheduling in Cloud Computing Environments." *Procedia Computer Science*. 65 : 920-929.

[59] Al-maamari, A. and Omara, F.A. 2015. "Task Scheduling Using PSO Algorithm in Cloud Computing Environments." *International Journal of Grid Distribution Computing*. 8(5) : 245-256.

[60] Basu, S. Karuppiah, Selvakumar, M. K. Li, K. C. Islam, S. K. H. Hassan, M. M. and Bhuiyan, M. Z. A. 2018. "An intelligent/cognitive model of task scheduling for IoT applications in cloud computing environment." *Future Gener. Comput. Syst.* 88 : 254-261.

[61] Kruekaew, B. and Kimpan, W. 2020. "Enhancing of Artificial Bee Colony Algorithm for Virtual Machine Scheduling and Load Balancing Problem in Cloud Computing" *Int. J. Comput. Intell. Syst.* 13(1) : 496-510.

[62] He, H. Xu, G. Pang, S. and Zhao,Z. 2016. "AMTS: Adaptive multi-objective task scheduling strategy in cloud computing" *China Commun.* 13(4) : 162-171.

[63] Reddy, G. N. and Kumar, S. P. 2017. "Multi Objective Task Scheduling Algorithm for Cloud Computing Using Whale Optimization Technique." pp. 286–297. in **Proceedings of International Conference Conf. Next Gener. Comput. Technol.** Singapore: Springer.

[64] Alsadie, D. 2021. "A Metaheuristic Framework for Dynamic Virtual Machine Allocation With Optimized Task Scheduling in Cloud Data Centers." *IEEE Access*. 9 : 74218-74233.

[65] Z. Amini, M. Maeen, and M. R. Jahangir, "Providing a load balancing method based on dragonfly optimization algorithm for resource allocation in cloud computing," *Int. J. Networked Distrib. Comput.,* vol. 6, no. 1, pp. 35-42, 2018, doi: 10.2991/ijndc.2018.6.1.4.

[66] S.H.H. Madni, M.S.A. Latiff, J. Ali, and S.M. Abdulhamid, "Multi-objective-oriented cuckoo search optimization-based resource scheduling algorithm for clouds," *Arab. J. Sci. Eng.*, vol. 44, no. 4, pp.3585-3602, 2019, doi: 10.1007/s13369-018-3602-7.

เอกสารอ้างอิง (ต่อ)

[67] S. Pang, W. Li, H. He, Z. Shan and X. Wang, "An EDA-GA Hybrid Algorithm for Multi-Objective Task Scheduling in Cloud Computing," in *IEEE Access*, vol. 7, pp. 146379-146389, 2019, doi: 10.1109/ACCESS.2019.2946216.

[68] P. Neelima, and A.R.M. Reddy, "An efficient load balancing system using adaptive dragonfly algorithm in cloud computing," *Cluster Comput.,* vol. 23, pp. 2891–2899, 2020, doi: 10.1007/s10586-020-03054-w.

[69] M. Gamal, R. Rizk, H. Mahdi, and B. E. Elnaghi, "Osmotic Bio-Inspired Load Balancing Algorithm in Cloud Computing," in *IEEE Access*, vol. 7, pp. 42735-42744, 2019, doi: 10.1109/ACCESS.2019.2907615.

[70] Butt, U. A. Mehmood, M. Shah, S. B. H. Amin, R. Shaukat, M. W. Raza, S. M. Suh, D. Y. and Piran, M. J. 2020. "A Review of Machine Learning Algorithms for Cloud Computing Security" *Electronics*. 9.

[71] Jena, U. K. Das, P. K. and Kabat, M. R. "Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment." *J. King Saud Univ. - Comput. Inf. Sci.,* to be published.

[72] Caviglione, L. Gaggero, Paolucci, M. M. and Ronco, R. 2021. "Deep reinforcement learning for multi-objective placement of virtual machines in cloud datacenters." *Soft. Comput.* 25 : 12569-12588.

[73] Abdelsamea, A. El-Moursy, A. A. Hemayed, E. E. and Eldeeb, H. 2017. "Virtual machine consolidation enhancement using hybrid regression algorithms." *Egypt. Inf. J.* 18(3) : 161-170.

[74] Kimpan, W. and Kruekaew, B. 2016. "Heuristic task scheduling with artificial bee colony algorithm for virtual machines." pp. 281–286. in **Proceeding of 2016 Joint 8th International Conference on Soft Computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS)**. IEEE, Sapporo, Japan.

[75] Zhou, Y. and Huang, X. 2013. "Scheduling workflow in cloud computing based on ant colony optimization algorithm." pp. 57–61. in **Proceeding of 2013 6th International Conference on Business Intelligence and Financial Engineering,** IEEE, Hangzhou, China.

เอกสารอ้างอิง (ต่อ)

[76] Al-Olima, H.S. Alam, M. Green, R. and Lee, J.K. 2015. “Cloudlet scheduling with particle swarm optimization.” pp. 991–995. in **Proceeding of 2015 Fifth International Conference on Communication Systems and Network Technologies**, IEEE, Gwalior, India.

[77] Saleh, H. Nashaat, H. Saber, W. and Harb, H.M. 2018. “IPSO task scheduling algorithm for large scale data in cloud computing environment.”, *IEEE Access.* 7 : 5412–5420.

[78] Kruekaew, B. and Kimpan, W. 2022. “Multi-Objective Task Scheduling Optimization for Load Balancing in Cloud Computing Environment Using Hybrid Artificial Bee Colony Algorithm With Reinforcement Learning.”, *IEEE Access*. 10 : 17803-17818. doi: 10.1109/ACCESS.2022.3149955.

ภาคผนวก

ภาคผนวก
งานวิจัยที่ตีพิมพ์

ATLANTIS PRESS

International Journal of Computational Intelligence Systems
Vol. **13(1)**, 2020, *pp.* 496–510
DOI: https://doi.org/10.2991/ijcis.d.200410.002; ISSN: 1875-6891; eISSN: 1875-6883
https://www.atlantis-press.com/journals/ijcis/

Regular paper

Enhancing of Artificial Bee Colony Algorithm for Virtual Machine Scheduling and Load Balancing Problem in Cloud Computing

Boonhatai Kruekaew, Warangkhana Kimpan*

Department of Computer Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand

ARTICLE INFO

Article History
Received 01 Oct 2019
Accepted 04 Apr 2020

Keywords
Artificial bee colony algorithm
Cloud computing
Scheduling algorithms
Load balance
Resource management
Distribution

ABSTRACT

This paper proposes the combination of Swarm Intelligence algorithm of artificial bee colony with heuristic scheduling algorithm, called Heuristic Task Scheduling with Artificial Bee Colony (HABC). This algorithm is applied to improve virtual machines scheduling solution for cloud computing within homogeneous and heterogeneous environments. It was introduced to minimize makespan and balance the loads. The scheduling performance of the cloud computing system with HABC was compared to that supplemented with other swarm intelligence algorithms: Ant Colony Optimization (ACO) with standard heuristic algorithm, Particle Swarm Optimization (PSO) with standard heuristic algorithm and improved PSO (IPSO) with standard heuristic algorithm. In our experiments, CloudSim was used to simulate systems that used different supplementing algorithms for the purpose of comparing their makespan and load balancing capability. The experimental results can be concluded that virtual machine scheduling management with artificial bee colony algorithm and largest job first (HABC_LJF) outperformed those with ACO, PSO, and IPSO.

1. INTRODUCTION

Artificial Bee Colony (ABC) is a Swarm Intelligent (SI) algorithm inspired by the foraging behavior of honey bees proposed by Karaboga in 2005 [1,2] and it is modified from Bee Colony Optimization (BCO) that was proposed for the first time in 2001 [3]. The foraging habits of honey bees are foraging and waggle dancing. The main idea is to create the multi agent system which is the colony of artificial bees to be able to efficiently solve hard combinatorial optimization problems. Therefore, ABC has been successfully employed in optimizations problems like data-mining problem [4,5], job shop scheduling [6,7], binary optimization [8–10], travelling salesman [11], biochemical networks [12], engineering optimization [13,14], image processing [15–17], as well as scheduling problem in cloud computing [18–21].

Cloud computing system [22,23] provides computing services via the Internet which will be operated upon user's requests. The requests are to manage the resources and services used through the system software. The user can control the amount of resources used such as CPU or RAM to an extent without having knowledge about the service types [24]. As cloud computing is based on virtualization technology that has no restriction on computing resources, it has a great advantage for service providers in terms of simplicity, energy saving, and low resource management cost. Cloud computing uses a virtualization technique to create Virtual Machines (VMs) [25,26]. It uses software to make a computer work like a system of multiple computers (VMs). VMs can replace a physical computer server and other necessary physical resources as they have their own independent virtual resources [27].

Tasks scheduling is a challenging problem and known to be NP-complete problem [28,29]. If the same server is simultaneously being requested to be used by many users, the other servers will become idle. This is called load imbalance. This problem can be solved by an algorithm to schedule tasks properly prior to service processing on VMs. A system with a proper task scheduling algorithm can also provide an advantage of efficient use of the system resources, i.e., it can reduce waiting time for a service queue and distribute some tasks to other servers. The implementation of this algorithm is called load balancing.

The operations of cloud computing are quite similar to those of ABC algorithm as both can adapt themselves to their environment which is a characteristic of the foraging behavior of bees. Their objectives are also similar: the objective of cloud computing is to maximize throughput for a desired makespan while that of bee foraging behavior is to find a food source with a large amount of honey with the least amount of effort.

The main contributions of this paper are to propose the idea of applying ABC algorithm and a task scheduling heuristic to each VM. The algorithm will minimize the makespan or the overall processing time of tasks and provide load balancing in cloud computing. Moreover, we focused on cloud computing within both homogeneous and heterogeneous environments and observed how

*Corresponding author. Email: warangkhana.ki@kmitl.ac.th

good the HABC algorithm works when the number of tasks in the system was varied and the type of datasets was changed. It is found that this algorithm performed the task scheduling better than ACO, PSO, and IPSO.

The paper is structured as follows: the related works, including task scheduling and load balancing in cloud computing and the ABC algorithm are described in Section 2. Adoption of a scheduling heuristic and load balancing algorithm inspired by ABC for cloud computing is proposed in Section 3. The experimental settings and results are discussed in Section 4. Finally, Section 5 presents the conclusion.

2. BACKGROUND AND RELATED WORK

2.1. Task Scheduling and Load Balancing in Cloud Computing

Cloud computing system creates a shared network resource and service with the user. Due to its diversified, dynamic, and flexible nature, different resources and services offering to different users, are the advantages of cloud computing. It provides a service upon user's request [24,30]. Cloud service providers offer various cloud services to user [22,23]. Resource management in cloud computing varies depending on the kind of user's request.

Heterogeneity in cloud computing depends on the infrastructure and service provided by providers. In case of providing the system with the same infrastructure to software packages and resources, the environment in the system will be called homogeneous cloud environment [31], while the environment in a system with different resources are provided by different providers, software packages by some other, infrastructure by yet another will be called heterogeneous cloud environment [32].

Over the last few years, exhaustive researchers have proposed several different task scheduling algorithms which run under the cloud computing resource utilization. Task scheduling algorithms mostly aimed to balance the workload that users requested for within the limitation of the system resources and to increase the efficiency of cloud computing. For example, Chase *et al.* [33] presented implementation of an architecture for resource management for large server clusters in order to provision server resources for co-hosted services and improve the energy efficiency of server clusters by dynamically resizing the active server set.

In 2009, D. Kusic *et al.* [34] presented a lookahead control scheme to solve a dynamic resource provisioning framework for a multi-objective optimization in a virtualized server environment. The technique can approach accounts for the switching costs incurred while provisioning VMs and explicitly encodes the corresponding risk in the optimization problem.

In 2011, D. Minarolli and B. Freisleben [35], proposed a scheme that dynamically allocates the resources to VMs. Such that quality of service constraints is satisfied, and its operating costs are minimized by a two-tier resource management approach based on adequate utility functions. Later in 2012, A. Gorbenko and V. Popov [36] proposed a logical model to solve the task-resource scheduling problem that minimized the resource and cost by applying algorithms for the satisfiability problem (SAT).

In 2013, S. Son *et al.* [37] proposed a Service Level Agreement (SLA) based on cloud computing framework to facilitate resource allocation considering the workload and geographical location of distributed data centers. This approach was suggested to tackle using the automated SLA negotiation mechanism and a Workload and Location-aware Resource Allocation scheme (WLARA) supports providers in earning higher profits. In 2015, A. Singh and K. Dutta [38], presented the resource allocation algorithm by Analytic Hierarchy Process (AHP) with a pairwise comparison matrix technique. This proposed technique made a better resource allocation than other algorithms. In 2017, Ullah *et al.* [39] presented a real-time resource usage prediction system which took real-time utilization of resources and fed utilization values into several buffers based on the resource types and time span size. Scheduling problem have been also discussed in different contents as in Refs. [40–42].

2.2. Related Work about Bioinspired and Meta-Heuristics Algorithms to Solve Cloud Computing

The nature-inspired metaheuristics algorithms attempt to attain the global optimal solution by examining the most suitable locations in the search space domain, based on the natural mechanism. The methods are different depending on type of algorithms, because some algorithms are suitable in solving specific type of problems but not suitable for others. This is due to the randomization or stochastic strategies that are used in solving those NP-Hard problems.

There are many previous works had proposed many metaheuristic-based approaches to scheduling workflow applications [43]. In particular, Genetic Algorithms (GA) [44,45], Particle Swarm Optimization (PSO) algorithms [46–49], Cuckoo Search (CS) algorithm [50–54], and Ant Colony Optimization (ACO) algorithm [55–56] have been used for scheduling workflows. For instance, Dasgupta *et al.* [45] proposed GA which is a balance the load of the cloud and minimizing the make span of a given tasks set. In 2010, S. Pandey *et al.* [46] proposed a PSO based heuristic to schedule applications to cloud resources that considers both computation cost and data transmission cost.

In 2019, H. Saleh *et al.* [49] proposed a cloud task scheduling with improved PSO (IPSO) algorithm that minimizes makespan and uses the resources in the system to the best advantage for a large number of tasks. T. P. Jacob and K. Pradeep [54] proposed hybrid algorithm is called as, CS algorithm, and particle swarm optimization (CPSO) for multi-objective-based task scheduling. They focused on the comparison of processing in terms of minimization of makespan, cost, and deadline violation rate in the heterogeneous cloud environment.

Many researchers had proposed approaches using metaheuristic-based to QoS-aware cloud service composition and cloud migration problem. In 2010, W. Li and H. Yan –xiang [57] developed a Chaos Particle Swarm Optimization (CPSM) method and provided a novel selection algorithm based on global QoS optimizing to solve QoS-aware service composition.

In 2014, Wang *et al.* [58] presented a modified GA to support data-intensive service composition. The research idea is to reduce the cost of service composition that involves large amounts of data transfer, data placement, and data storage.

In 2016, L. Wang and J. Shen [59] presented ant colony system for the dynamic service composition problem multi-phase, multi-party negotiation protocol. The ant colony system was applied to select services with the best or near-optimal utility outputs. Ferdaus *et al.* [60] proposed a novel ACO-based Migration impact-aware Dynamic VM Consolidation (AMDVMC) algorithm in order to solve the Multi-objective, Dynamic VM Consolidation Problem (MDVCP) in computing clouds for minimizing data center resource wastage, power consumption, and overall migration overhead due to VM consolidation.

In 2019, S. K. Gavvala *et al.* [61] proposed a novel Eagle Strategy with Whale Optimization Algorithm (ESWOA) with eagle strategy to balance the exploration and exploitation in QoS-aware Cloud service composition.

2.3. Overview of the ABC Algorithm

ABC algorithm [1,2,62] is an optimization technique for finding an optimal solution. It was inspired by the foraging behavior and collective work of different kinds of bees. In a bee colony, there are 3 types of artificial bees: Scout bees with a duty to randomly search for new food sources, Employed bees that go to the food sources and come back to dance in the hive according to the position and the amount of the food to Onlooker bees that are waiting in the hives, and the last one is Onlooker bees that calculate the fitness value of each food source and choose the optimal source for the Employed bees to go to and collect the food.

If a food source was chosen and all the food was collected, the Employed bees which are the owners of the food source will turn into Scout bees and randomly search for new food sources again. In nature, bees communicate the quality of food sources at a dance floor. They dance a "Waggle Dance" to inform others about the direction to go, the distance from a food source, and the amount of honey of a food source. The Onlooker bees compare all of the food sources and determine the best one. The steps in the ABC algorithm are shown in Figure 1.

ABC algorithm was used in many fields such as job shop scheduling [6,7], binary optimization [8–10], travelling salesman [11], digital signal processing [63], and so on. For example, Mizan *et al.* [18] presented job scheduling in Hybrid cloud by modifying Bee Life algorithm and Greedy algorithm to achieve an affirmative response from the end users and utilize the resources. Accordingly, the objective of this study is to optimize task scheduling, resource allocation, and load balancing using applying Heuristic Task Scheduling with Artificial Bee Colony (HABC) based on the proposed reducing or minimizing makespan in cloud computing environment. This research focused on both cases of homogeneous and heterogeneous environments and investigated how efficient the HABC algorithm performed when the tasks in the system were increased and the data distribution was changed.

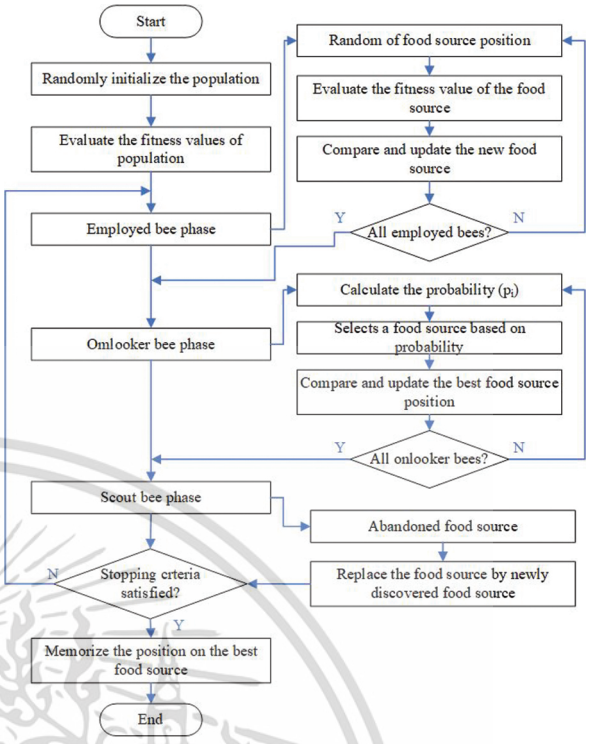


Figure 1 Flowchart of Artificial Bee Colony (ABC) algorithm

3. LOAD BALANCING ALGORITHM INSPIRED BY ARTIFICIAL BEE COLONY AND HEURISTIC SCHEDULING ALGORITHM

Cloud computing utilizes different resources depending on the types of user's request. Moreover, fulfilment of the request of service depends on workload of servers. If a cloud system has an efficient task scheduling, it will help reduce the makespan and balance the workload among the servers. In Ref. [19] and Ref. [20], ABC algorithm was used to manage VMs scheduling in cloud computing within a homogeneous environment. However, since the specifications of current servers are very different, ABC algorithm with heuristic task scheduling was also applied to a heterogeneous environment in this study. The two goals of the proposed method were to obtain a good workload balance for maximum productivity and minimize total makespan. This section describes the heuristic scheduling algorithm and the proposed scheduling algorithm.

3.1. Heuristic Scheduling Algorithm

Heuristic method is an approach to problem solving using rules such as priority rule or using a random method to find an optimal solution for a problem. A heuristic method may provide a good result, but it cannot guarantee an optimal solution. Nevertheless, it is simple to use. A heuristic task scheduling with priority as a criterion for selecting a process in a system was used

in this study because it is a basic method for scheduling tasks [64,65]. We selected the following 3 simple priority criteria for investigation [66]:

- First Come First Serve (FCFS): is a task scheduling scheme that processes the first job that the user has requested first.
- Smallest Job First (SJF): is a task scheduling scheme that processes with the smallest job first
- Largest Job First (LJF): is a task scheduling scheme that processes the largest job first.

3.2. Proposed Method

Let $VM = \{vm_1, vm_2, vm_3, \ldots, vm_m\}$, variables used in this study and their definitions derive from Table 1. Let the system be set to run in an uninterruptible manner, i.e., nonpreemptive. Before the ABC algorithm was run, the tasks were arranged into 3 different arrangements by using the Heuristic algorithms. The best arrangement would use the least computation time when the tasks were processed by the ABC algorithm as mentioned above. The ABC algorithm then scheduled the tasks to access the VMs in the following steps:

1. *First step:* the number of bees (n) in the population was specified. They were assigned randomly to go to all of the different food sources (m) that represented the VMs and their fitness values were calculated, therefore initializing the VMs as shown in Algorithm 1.

Algorithm 1: : Initialization

1. **For** i = 1 **to** n
2. Send the population of bees into the system to find appropriate VMs by Random search.
3. Calculate the fitness of each VM by using (1)

$$C_j = Pe_j \times Mi_j + Bw_j \tag{1}$$

where j = index of the VM that the i^{th} bee found.
4. **End For**

2. *Second step:* a different default value was assigned to each food source according to the specifications of the VM. Bees were grouped into 3 types: Scout Bees, Employed Bees, and Onlooker Bees. A Scout Bee found the initial position of a food source. An Employed Bee went to the food source, recalculated and updated the fitness value of food source. An Onlooker Bee decided which food source was the best food source. The operation of an employed bee is presented in Algorithm 2.

Algorithm 2: Employed bee phase

1. **For** i = 1 **to** n
2. Employed bees are sent randomly to food sources (VMs in cloud computing).
3. Fitness of each VM is calculated based on (2)

$$F_{ij} = \frac{\sum_{i=1}^{n} Tl_{ij}}{Evaluate\ capacity\ of\ VM_j\ \left(C_j\right)} \tag{2}$$

4. Update the fitness value
5. **End For**

3. *Third step:* the Employed bees have searched around for food sources, they brought back information about the food sources to the Onlooker bee. The Onlooker bee then recalculated the fitness values of the food sources. The operation of the onlooker bee is shown in Algorithm 3.

Algorithm 3: Onlooker bee phase

1. The onlooker bee chooses the first m food sources (VM) with the highest fitness values and perform "Neighborhood Search" around the food sources
2. An nsp number of employed bees are sent to bring back new positions of the first m food sources while a nep number of employed bees are sent to all food sources to bring back new positions of these food sources; all of them come back to relay the information to the onlooker bee
3. The onlooker bee calculates the new fitness values of the food sources according to (3),

$$fit_{ij} = \frac{\sum_{i=1}^{n} Tl_i + In_{length}}{Evaluate\ capacity\ of\ VM_j\ \left(C_j\right)} \tag{3}$$

where In_{length} is the length of the task that is waiting to access a VM at that time
4. The onlooker bee chooses the best food source (VM) and assigns a task to the VM

4. *Fourth step:* the employed bee that was the owner of the best food source was then transformed into a scout bee. The operation of a scout bee is shown in Algorithm 4.

Algorithm 4: Scout bee phase

1. **If** food source = null **then**
2. Send the previously transformed scout bee into the system to find an appropriate VM by Random search.
3. Calculate the fitness value of that VM by using (1)
4. **End If**

5. *Fifth step:* the overall operation of HABC algorithm is described in Algorithm 5.

Algorithm 5: Improved HABC Algorithm

Input: Dataset, Bee's Parameters;
Output: Minimum makespan;
1. Initialize the individual in the population.
2. Set the default values of the food sources (VM) by using (2).
3. *Initialization* (Algorithm 1)
4. **Repeat**
5. *Employed bee phase* (Algorithm 2)
6. *Onlooker bee phase* (Algorithm 3)

7. Calculate the load balance value of the system after task scheduling on VMs. It is calculated by using Standard Deviation (S.D.) [67] as in (4) and (5),

$$S.D. = \sqrt{\frac{1}{n}\sum_{j=0}^{n}\left(X_j - \overline{X}\right)^2} \tag{4}$$

$$imbalance = \frac{X_{max} - X_{min}}{X_{avg}} \tag{5}$$

The processing time for each VM (X_j) is calculated according to (6),

$$X_j = \frac{\sum_{i=1}^{k} task\_length_i}{capacity_j} \tag{6}$$

The average processing time for running a VM in the system ($\overline{X}$) is calculated according to (7),

$$\overline{X} = \frac{\sum_{j=1}^{n} X_j}{n} \tag{7}$$

If S.D. $\leq \overline{X}$, it means that the system is in a balanced condition whereas if S.D. $\leq \overline{X}$, it means that the system is in a state of imbalance.
8. *Scout bee phase* (Algorithm 4)
9. **Until** (the termination condition is met)

Table 1 Variables used in this study and their definitions.

Symbol	Definition
m	The number of virtual machines (VMs)
$VM = \{vm, vm_2, vm_3, \ldots, vm_m\}$	The set of VM
K	The total number of tasks the system has to perform
$Task = \{t_1, t_2, t_3, \ldots, t_K\}$	The set of tasks
N	The total number of bees performing in the algorithmic procedure
C_j	The performance of the j^{th} virtual machine
Pe	The number of processors in the VM
Mi	Million Instructions Per Second (MIPS)
Bw	Bandwidth of VM
F	Fitness
Tl	The length of task in MI[a]
P_i	Probability that the i^{th} food source is good which depends on its fitness value, $p_i = \frac{fitness_i}{\sum_{i=1}^{NS} fitness_i}$, where NS is the size of food sources

4. EXPERIMENTAL RESULTS

4.1. System Environment Parameter Settings

The algorithms: HABC, ACO, PSO, and IPSO described were run using CloudSim-3.0.1 [68,69] tools on a personal computer. The experiments were performed on 10 data centers and 50 VMs. In the experiments, we focused on both kinds of environments: homogeneous and heterogeneous environments in cloud computing.

For each VM in cloud computing within a homogeneous environment, the following parameters were set: Instructions Per Second in millions (MIPS is a measurement of the processing speed of a computer) of Processing Element (PE) were fixed at 9726; Number of PE per VM was fixed at 1; and RAM used was fixed at 1024 Megabytes (MB).

For each VM in cloud computing within a heterogeneous environment, the following parameters were set: MIPS were in the range of 5000–9726; Number of PE per VM was fixed at 2; RAM used was set in the range of 512–4096 MB; and Bandwidth was set in the range of 1000–10000.

4.2. The Datasets Used in the Experiments

To evaluate the effectiveness of the proposed approach, twelve datasets, shown in Table 2, were used. Those datasets were of 4 types. The set D_1–D_3 of the first type was a set of random information, i.e., there are no definite terms, no specific format, and no specific distribution of information. The set D_4–D_6 of the second type was a set of random data with normal distribution. The set D_7–D_9 of the third type was a set of random data with a Right-Skewed Distribution [70]. Lastly, the set D_{10}–D_{12} of the fourth type was a set of random data with a Left-Skewed Distribution. Each type of datasets had a different range of job size and each dataset comprised a number of groups of 100, 200, 300, ⋯, 1500 submitted tasks.

4.3. Parameters Settings for the Proposed Algorithm and the Comparing Algorithms

In these experiments, we have determined the parameters of the population and various conditions according to the related works, which are referred to HABC [20], ACO [55,56], PSO [48], and IPSO [49]. It is well known that these parameters are highly effect on the algorithms. The determination of parameters is varied according to the dimension of problems or characteristics. One reason that we need tuning the parameters [20] is in order to get the suitable parameters for the pattern of problems and datasets. However, we do not claim that the proposed algorithm or the specified parameters outperform other algorithms for all types of problems and datasets. Therefore, tuning parameters is the best way to get the suitable parameters to solve problems.

Table 2 The datasets used in the experiments.

Type of Dataset	Dataset Name	Length of Task (MI[a])
Random	D_1	[5000, 20000]
	D_2	[15000, 35000]
	D_3	[25000, 45000]
Normal distribution	D_4	[5000, 20000]
	D_5	[15000, 35000]
	D_6	[25000, 45000]
Right-skewed distribution[b]	D_7	[5000, 20000]
	D_8	[15000, 35000]
	D_9	[25000, 45000]
Left-skewed distribution[c]	D_{10}	[5000, 20000]
	D_{11}	[15000, 35000]
	D_{12}	[25000, 45000]

(a) MI is Million Instruction. (b) Positively Skewed Curve. (c) Negatively Skewed Curve.

The experiments had been performed to determine the optimal bee parameters for the proposed algorithm [20] which are shown in Table 3. In addition, the scheduling performance of the proposed algorithm was also compared with those of ACO [55,56], PSO [48], and IPSO [49] that used the parameter settings as shown in Table 3.

Those optimal settings were based on the original papers. The number of iterations that each algorithm would be run in the experiment was 20.

4.4. Experimental Results and Discussions

In one of our previous papers [20], the performances of the proposed algorithm in combination with 3 heuristics-HABC_FCFS, HABC_LJF, and HABC_SJF-were compared with those of the heuristics alone-FCFS, LJF, and SJF. The results of previous experiments showed that when the largest job was considered first (HABC_LJF), its scheduling performance was better than those of HABC_FCFS and HABC_SJF.

The HABC_LJF also balanced the load of submitted tasks with minimal makespan. In this paper, we used 2 VM environments: heterogeneous and homogeneous to evaluate the performance of HABC_LJF method on different datasets and to compare it with other scheduling algorithms. The number of VMs was fixed throughout the experiment but the number of tasks was increased in 100 incremental steps from 100 to 1,500 tasks.

In the evaluation of the performance of HABC_LJF algorithm in cloud computing, we divided our experiment into two parts. In the first part, the datasets of various types as described in Table 3 were used. In the second part, the performance of the heuristic task scheduling with ABC in terms of load imbalance was evaluated. In the first part, the performance evaluation was performed on 4 cases according to the type of datasets used. The following are the results of the 4 cases for evaluation of the makespan.

Case 1: *Random datasets*

The makespan of HABC_LJF running on random datasets D_1–D_3 was compared with those of ACO, PSO, and IPSO algorithms. The number of groups of submitted tasks was varied in each run from 100 to 1500 submitted tasks in an increment of 100 tasks. The average makespan for 300, 900, and 1,500 tasks in cloud computing within homogeneous and heterogeneous environments are shown in Tables 4 and 5, respectively. Actually, a higher number of tasks were run but the results are not shown here due to lack of space.

According to Table 4, HABC_LJF performed the best as the number of tasks was increased. Moreover, the minimum makespan, the maximum makespan, and the average makespan, all three of them were the lowest. The performance of HABC_LJF was outstanding in the homogeneous environment.

In addition, in order to observe the effect of the range of job size on performance scalability of the system, we evaluated the average makespan with three datasets that had different ranges of job size (D_1–D_3). The results show that the HABC_LJF algorithm also outperformed the others for all size ranges tested in the homogeneous environment.

The results for the average makespan of HABC_LJF algorithm and the other algorithms with an increasing number of submitted tasks in cloud computing within a heterogeneous environment when used with random datasets are shown in Table 5. The performance of HABC_LJF was the best because its minimum makespan, maximum makespan, and average makespan were mostly the lowest in cloud computing within a heterogeneous environment. Furthermore, when datasets with different ranges of job size were used, HABC_LJF still outperformed the others in the same way that it did in homogeneous environment.

Case 2: *Random data with a normal distribution*

Table 6 shows the makespan of every algorithm tested with datasets (D_4–D_6) : random datasets with normal distribution. The results showed that the HABC_LJF used the lowest minimum makespan, the lowest maximum makespan, and the lowest average makespan than the other algorithms did in the homogeneous environment.

The makespan of the algorithms working within a heterogeneous environment when random datasets with normal distribution were used are shown in Table 7. The results show that for the datasets that had different ranges of job size, HABC_LJF outperformed the others in the same way as they did in a homogeneous environment except for the case of dataset D_4 and D_6. For the case of dataset D_4 with the number of submitted tasks equaled to 900, the ACO_LJF gave the lowest minimum makespan. Meanwhile, the D_6 dataset with 900 submitted tasks in which ACO_LJF used the lowest minimum makespan and ACO_SJF used the lowest maximum

Table 3 The values of the parameters used in the proposed algorithm and in the other comparison algorithms.

HABC [20]		ACO [55,56]		PSO [48]		IPSO [49]	
Parameter	**Values**	**Parameter**	**Values**	**Parameter**	**Values**	**Parameter**	**Values**
Number of scout bees (n)	960	Ants	50	Number of particles	100	Population size	100
Number of sites selected out of n visited sites (m)	96	Maximum probability of trial	0.98	Weight (w)	0.9	Weight (w_{min}, w_{max})	0.1, 0.9
Number of best sites out of m selected sites (e)	1	Local search probability	0.01	Acceleration factor (C_1)	1.49445	Acceleration factor (C_1)	1.49445
Number of bees recruited for searching best e sites (nep)	768	Evaporation rate	0.01	Acceleration factor (C_2)	1.49445	Acceleration factor (C_2)	1.49445
Number of bees recruited for searching other (m-e) selected sites (nsp)	192	Max number of iterations	1000	Max number of iterations	1000	Maximum iterations	1000
Max number of iterations	1000					K	5

HABC, Heuristic Task Scheduling with Artificial Bee Colony; ACO, Ant Colony Optimization; PSO, Particle Swarm Optimization; IPSO, improved PSO.

Table 4 The comparison of the average makespan of the systems with HABC, ACO, PSO, and IPSO in a homogeneous environment with datasets $D_1 - D_3$.

Dataset	Task	Variable	HABC_LJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_FCFS	IPSO_LJF	IPSO_SJF
D_1	300	Min	**8.623**	9.600	9.023	9.982	25.982	25.030	25.707	29.578	27.448	27.979
		Max	**8.643**	10.592	9.343	11.017	29.017	26.329	26.408	32.381	31.162	34.342
		Avg	**8.633**	10.227	9.225	10.324	26.988	25.563	25.876	30.626	28.766	30.273
	900	Min	**25.015**	26.352	25.739	26.702	83.733	81.206	83.467	107.237	103.885	105.384
		Max	**25.262**	27.369	26.042	27.561	84.777	96.767	86.560	113.512	108.464	107.781
		Avg	**25.073**	26.819	25.881	26.976	84.165	89.705	84.255	109.620	105.725	106.302
	1500	Min	**42.275**	43.641	42.892	44.050	151.965	153.637	153.482	215.075	211.427	303.934
		Max	**42.614**	45.020	43.523	45.058	164.660	156.171	157.010	229.624	315.327	319.373
		Avg	**42.409**	44.233	43.228	44.641	158.947	154.736	154.529	222.093	262.953	310.891
D_2	300	Min	**16.308**	18.685	18.036	19.148	36.465	32.567	33.462	37.476	35.041	37.075
		Max	**16.341**	19.719	18.288	20.252	39.547	34.134	35.205	39.296	42.372	39.112
		Avg	**16.319**	19.258	18.161	19.646	38.194	32.993	33.906	38.645	38.841	37.428
	900	Min	**48.340**	50.969	50.150	51.526	108.373	105.943	106.805	129.830	128.848	127.957
		Max	**48.907**	52.823	51.427	53.980	109.092	108.074	108.557	137.917	132.341	130.492
		Avg	**48.475**	51.795	50.353	52.154	108.704	106.622	107.725	133.191	130.587	129.206
	1500	Min	**80.237**	82.888	81.959	83.918	192.893	189.624	194.519	250.603	245.937	248.938
		Max	**81.623**	84.951	83.743	86.442	193.491	192.754	198.249	263.329	256.496	254.803
		Avg	**80.516**	84.122	82.818	85.263	193.206	191.226	196.048	255.369	251.500	251.033
D_3	300	Min	**22.229**	25.645	24.983	26.244	40.877	38.553	39.606	45.246	45.293	46.121
		Max	**22.262**	27.500	25.278	26.867	42.182	41.206	41.803	56.463	50.604	50.163
		Avg	**22.244**	26.378	25.126	26.529	41.230	39.237	40.155	49.921	47.083	47.706
	900	Min	**67.077**	70.642	70.006	71.203	127.109	125.271	125.686	158.775	158.025	160.241
		Max	**67.245**	73.066	72.770	74.383	128.451	128.572	128.420	176.779	174.420	169.610
		Avg	**67.188**	71.676	70.506	72.115	127.780	126.110	126.468	169.075	165.177	164.849
	1500	Min	**111.770**	116.333	114.901	116.010	223.859	222.317	223.264	290.299	290.595	290.642
		Max	**112.050**	119.118	118.887	119.478	227.468	225.851	225.024	304.952	289.595	300.847
		Avg	**111.870**	117.533	116.918	118.256	225.457	223.576	224.219	296.090	293.595	294.004

HABC, Heuristic Task Scheduling with Artificial Bee Colony; ACO, Ant Colony Optimization; PSO, Particle Swarm Optimization; IPSO, improved PSO; FCFS, First Come First Serve; SJF, Smallest Job First; LJF, Largest Job First.

Table 5 The comparison of the average makespan of the systems with HABC, ACO, PSO, and IPSO in a heterogeneous environment with datasets $D_1 - D_3$.

Dataset	Task	Variable	HABC_LJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_FCFS	IPSO_LJF	IPSO_SJF
D_1	300	Min	**7.320**	7.982	7.500	8.292	24.207	23.707	25.749	27.498	26.623	27.055
		Max	**7.368**	9.803	8.181	9.925	27.171	24.370	26.155	30.966	29.537	27.616
		Avg	**7.345**	8.817	7.765	9.086	25.610	23.996	26.007	28.792	27.144	27.322
	900	Min	**20.601**	21.133	20.605	21.707	80.445	79.467	81.114	101.135	98.005	99.690
		Max	**20.865**	23.311	21.808	22.592	82.856	81.043	88.297	106.207	111.467	101.623
		Avg	**20.715**	22.256	20.941	22.146	81.698	79.943	82.409	102.858	100.261	100.695
	1500	Min	**34.213**	34.976	34.218	35.310	148.688	148.288	148.757	216.645	213.903	219.464
		Max	**34.717**	37.461	35.203	35.970	152.961	150.648	151.124	247.682	256.862	247.018
		Avg	**34.450**	36.114	34.700	35.630	150.867	149.111	149.999	227.404	232.807	231.975
D_2	300	Min	**14.595**	15.385	14.935	15.933	32.330	30.390	33.420	32.324	32.507	34.153
		Max	**14.687**	18.406	15.641	17.333	36.045	30.760	33.823	34.935	34.675	35.080
		Avg	**14.615**	16.978	15.316	16.839	33.353	30.545	33.666	33.442	33.155	34.533
	900	Min	**40.299**	40.305	40.486	41.219	98.444	96.444	98.889	118.667	118.246	119.673
		Max	**40.706**	44.432	41.102	43.978	101.081	97.356	100.745	131.408	123.997	124.640
		Avg	**40.439**	42.816	40.821	42.402	99.779	96.823	99.582	124.194	119.795	121.197
	1500	Min	**65.784**	66.160	66.063	66.240	175.483	175.397	169.006	229.331	230.574	233.232
		Max	**67.162**	69.771	67.677	68.459	181.691	178.763	204.867	261.110	261.855	237.858
		Avg	**66.243**	67.947	66.418	67.835	177.890	177.211	180.840	239.835	237.419	234.780
D_3	300	Min	**20.374**	21.310	20.541	22.037	36.164	35.042	38.526	38.310	36.574	40.152
		Max	**20.595**	25.453	21.479	23.777	41.005	35.376	39.248	42.268	39.352	40.698
		Avg	**20.412**	23.130	21.084	22.705	38.605	35.183	38.678	39.821	38.069	40.422
	900	Min	**56.053**	56.317	56.796	57.418	114.889	108.005	109.332	128.071	125.720	129.099
		Aax	**56.986**	61.682	57.280	59.408	118.975	109.462	112.828	135.651	129.380	131.149
		Avg	**56.632**	58.188	57.014	58.687	116.954	108.515	110.325	130.650	127.624	130.120
	1500	Min	**92.419**	93.852	93.374	93.751	195.333	196.762	191.022	264.640	244.018	245.887
		Max	**92.945**	96.593	93.358	94.951	222.115	219.735	195.878	316.860	278.675	251.534
		Avg	**92.624**	95.327	93.005	94.492	205.862	207.061	192.619	285.370	256.390	248.354

HABC, Heuristic Task Scheduling with Artificial Bee Colony; ACO, Ant Colony Optimization; PSO, Particle Swarm Optimization; IPSO, improved PSO; FCFS, First Come First Serve; SJF, Smallest Job First; LJF, Largest Job First.

Table 6 The comparison of the average makespan of the systems with HABC, ACO, PSO, and IPSO in a homogeneous environment with datasets D_4 – D_6.

Dataset	Task	Variable	HABC_LJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_FCFS	IPSO_LJF	IPSO_SJF
D_4	300	Min	**8.575**	9.883	9.111	10.170	25.540	24.501	25.472	29.994	28.758	28.635
		Max	**8.634**	11.052	9.501	10.740	26.333	24.940	26.308	32.600	31.167	29.165
		Avg	**8.584**	10.331	9.308	10.434	25.798	24.643	25.813	30.624	29.298	28.851
	900	Min	**25.582**	27.020	26.319	27.101	82.715	81.997	84.421	110.198	107.394	107.813
		Max	**25.867**	27.866	26.746	28.302	92.448	84.768	87.776	119.204	110.113	110.444
		Avg	**25.669**	27.400	26.511	27.644	84.544	83.432	85.733	113.127	108.915	109.174
	1500	Min	**41.993**	43.462	42.731	43.983	149.972	152.230	151.124	221.121	215.575	218.054
		Max	**42.246**	44.370	44.234	44.703	157.576	154.169	161.643	233.171	220.500	221.989
		Avg	**42.140**	43.799	43.177	44.364	152.614	153.111	153.875	225.417	217.941	219.824
D_5	300	Min	**16.182**	18.335	17.793	18.982	34.274	32.607	33.625	37.653	38.349	37.446
		Max	**16.215**	20.528	19.076	20.624	35.227	34.275	34.922	42.583	40.628	37.899
		Avg	**16.194**	19.098	18.004	19.429	34.484	32.893	33.952	40.022	38.938	37.747
	900	Min	**48.367**	51.071	50.232	51.457	108.799	106.392	107.270	135.063	129.111	130.951
		Max	**48.499**	52.978	51.790	54.115	118.703	108.260	108.692	138.506	134.602	134.658
		Avg	**48.404**	52.034	50.572	52.473	110.707	107.140	107.825	136.418	132.357	132.098
	1500	Min	**80.594**	83.666	82.612	83.878	193.607	192.516	192.841	258.106	252.917	254.743
		Max	**80.808**	85.558	84.510	86.401	256.281	194.782	194.970	285.920	270.308	258.800
		Avg	**80.701**	84.506	83.578	85.431	207.942	193.393	193.883	263.625	258.098	256.448
D_6	300	Min	**22.266**	25.403	24.843	26.012	40.675	38.361	39.239	45.958	45.530	44.446
		Max	**22.322**	27.499	25.273	27.442	40.989	43.362	46.370	47.159	49.020	45.375
		Avg	**22.275**	26.316	25.094	26.585	40.832	39.170	41.054	46.464	46.335	44.698
	900	Min	**66.586**	70.473	69.570	70.620	125.921	123.478	123.851	151.630	150.560	149.820
		Max	**66.733**	72.472	71.977	73.627	126.799	126.650	127.080	157.917	157.273	151.973
		Avg	**66.675**	71.373	70.054	71.545	126.492	124.704	125.075	154.434	152.390	151.096
	1500	Min	**111.167**	115.941	114.366	115.366	223.019	221.505	222.173	287.264	281.794	285.184
		Max	**111.508**	117.362	117.450	118.750	226.545	259.360	224.472	297.683	292.236	293.692
		Avg	**111.291**	116.548	116.070	117.391	224.414	238.211	223.379	291.949	288.023	287.741

HABC, Heuristic Task Scheduling with Artificial Bee Colony; ACO, Ant Colony Optimization; PSO, Particle Swarm Optimization; IPSO, improved PSO; FCFS, First Come First Serve; SJF, Smallest Job First; LJF, Largest Job First.

Table 7 The comparison of the average makespan of the systems with HABC, ACO, PSO, and IPSO in a heterogeneous environment with datasets D_4 – D_6.

Dataset	Task	Variable	HABC_LJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_FCFS	IPSO_LJF	IPSO_SJF
D_4	300	Min	**7.347**	8.266	7.460	8.378	25.131	23.905	24.550	28.075	28.058	28.192
		Max	**7.536**	9.874	7.993	9.678	27.318	25.652	25.811	33.277	29.661	30.436
		Avg	**7.429**	9.157	7.711	9.006	25.850	24.791	24.849	30.191	28.811	28.953
	900	Min	**20.822**	21.427	20.973	22.206	79.248	81.988	77.327	98.817	96.358	98.507
		Max	21.869	23.342	**21.397**	23.313	81.374	85.940	78.608	103.967	98.902	101.255
		Avg	**21.215**	22.360	21.226	22.597	80.299	84.128	77.870	100.968	97.827	99.515
	1500	Min	**33.862**	34.144	33.987	34.943	141.381	140.255	140.594	199.534	195.573	197.259
		Max	**34.482**	36.491	35.618	35.978	162.631	153.213	144.570	211.162	201.047	202.690
		Avg	**34.210**	35.305	34.399	35.434	151.768	144.389	142.179	203.859	198.525	199.162
D_5	300	Min	**14.211**	14.876	14.837	16.105	25.131	23.905	24.550	33.591	32.701	34.567
		Max	**14.306**	18.346	15.362	17.862	27.318	25.652	25.811	37.458	33.676	35.061
		Avg	**14.261**	16.768	15.100	16.640	25.850	24.791	24.849	35.377	33.052	34.751
	900	Min	**40.087**	41.393	40.543	41.148	79.248	81.988	77.327	116.100	114.003	117.011
		Max	**40.735**	47.170	41.090	42.831	81.374	85.940	78.608	122.000	116.834	118.787
		Avg	**40.256**	43.287	40.754	42.407	80.299	84.128	77.870	118.567	115.730	117.796
	1500	Min	**66.087**	66.979	66.228	66.660	141.381	140.255	140.594	231.830	228.575	230.755
		Max	**66.948**	70.342	67.070	72.701	162.631	153.213	144.570	242.655	233.695	235.654
		Avg	**66.407**	68.998	66.679	68.511	151.768	144.389	142.179	236.277	231.274	233.031
D_6	300	Min	**20.300**	21.324	20.600	21.816	35.892	34.222	38.002	38.696	37.067	40.703
		Max	**20.606**	25.610	21.514	23.305	39.573	34.525	38.280	43.352	38.837	41.195
		Avg	**20.412**	23.303	21.077	22.851	37.991	34.350	38.112	40.338	38.347	40.964
	900	Min	55.891	**55.881**	56.112	57.728	107.681	104.625	108.412	129.248	127.836	130.677
		Max	56.877	60.508	**56.811**	59.122	109.710	106.115	110.034	135.833	131.669	132.884
		Avg	**56.185**	58.464	56.507	58.322	108.418	105.422	108.966	131.469	129.420	131.899
	1500	Min	**92.022**	93.339	92.170	93.697	188.914	188.526	190.508	251.495	249.411	250.938
		Max	**93.022**	98.200	93.025	94.655	194.341	191.091	192.220	264.148	253.773	256.640
		Avg	**92.032**	95.466	92.498	94.106	191.243	189.838	191.406	255.573	251.396	253.024

HABC, Heuristic Task Scheduling with Artificial Bee Colony; ACO, Ant Colony Optimization; PSO, Particle Swarm Optimization; IPSO, improved PSO; FCFS, First Come First Serve; SJF, Smallest Job First; LJF, Largest Job First.

makespan. However, this makespan was not very different from that used by HABC_LJF. When the makespan for all cases were considered, HABC_LJF still outperformed the other algorithms.

Case 3: *Random data with Right-Skewed Distribution*

Tables 8 and 9 show the results obtained from using datasets (D_7–D_9)—random datasets with Right-Skewed Distribution—in homogeneous and heterogeneous environments, respectively. The experimental results indicate that HABC_LJF used the lowest minimum makespan, the lowest maximum makespan, and the lowest average makespan than the other algorithms did in the homogeneous environment. The results were the same as in Case 1.

In the same direction, HABC_LJF outperformed the other two algorithms in a heterogeneous environment except when D_9 was used with the number of submitted tasks equaled to 900 and 1500 tasks. For the case of 900 tasks, ACO_FCFS used the lowest minimum makespan. For the case of 1500 tasks, ACO_LJF used the lowest maximum makespan. Nevertheless, in both cases, the minimum makespan used by ACO_FCFS and ACO_LJF were not very different from that used by HABC_LJF. Overall, HABC_LJF still outperformed them both.

Case 4: *Random data with Left-Skewed Distribution*

The makespan of HABC_LJF were compared with those of ACO, PSO, and IPSO algorithms running on Left-Skewed Distribution (D_{10}–D_{12}) datasets. Their minimum, maximum, and average makespan in cloud computing within homogeneous and heterogeneous environments are shown in Tables 10 and 11, respectively.

The results in Table 10 show that HABC_LJF outperformed the other three algorithms in a homogeneous environment. The results were the same as in other case.

In the same direction, HABC_LJF outperformed the other three algorithms in a heterogeneous environment except for one case: when using D_{11} dataset with 900 submitted tasks, the ACO_LJF gave the lowest maximum makespan but this makespan was not much different from that of HABC_LJF. When the makespan of all cases were considered, the HABC_LJF still outperformed the others.

In the second part, the performance of the HABC algorithm in terms of load imbalance was evaluated. Unbalanced workload for VMs can be avoided by balancing the workload during allocation. The datasets of various types were used as described in Table 3. Four cases of performance evaluation were considered according to the type of datasets used. The number of groups of submitted tasks was varied in each run from 100 to 1500 in an increment of 100 tasks. Performance evaluation was also done separately for the homogeneous and heterogeneous VM environments to obtain the average degree of load imbalance of 20 runs for each algorithm. The degrees of load imbalance in cloud computing within the homogeneous and heterogeneous environments are shown in Figures 2 and 3, respectively.

As shown in Figure 2, in order to consider the effect of different types of dataset on performance scalability of the system, we evaluated the average degrees of load imbalance with four datasets of different types: D_1, D_4, D_7, and D_{10}. The results show that

Table 8 The comparison of the average makespan of the systems with HABC, ACO, PSO, and IPSO in a homogeneous environment with datasets D_7 – D_9.

Dataset	Task	Variable	HABC_LJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_FCFS	IPSO_LJF	IPSO_SJF
D_7	300	Min	**8.173**	9.523	8.788	9.600	25.399	24.606	25.048	28.675	28.359	28.078
		Max	**8.248**	10.350	9.068	10.213	25.589	24.846	26.086	29.708	30.863	28.928
		Avg	**8.211**	9.951	8.950	9.904	25.524	24.696	25.554	29.090	29.149	28.515
	900	Min	**23.742**	25.072	24.179	25.155	83.196	81.680	81.820	106.287	104.397	105.338
		Max	**24.130**	26.153	24.727	26.494	83.696	82.620	83.303	110.536	106.107	109.230
		Avg	**23.874**	25.520	24.384	25.640	83.526	82.159	82.205	107.879	105.350	106.786
	1500	Min	**39.575**	41.225	40.088	41.291	151.844	149.901	150.912	227.688	211.238	214.100
		Max	**39.866**	42.121	40.807	42.272	152.875	152.081	152.960	239.956	228.046	230.281
		Avg	**39.679**	41.569	40.433	41.866	152.226	151.149	152.024	232.735	216.979	224.078
D_8	300	Min	**14.604**	16.839	16.258	17.553	32.596	32.379	23.083	37.881	37.521	36.366
		Max	**14.622**	19.080	16.541	18.576	33.470	34.339	23.167	39.898	38.762	37.016
		Avg	**14.613**	17.792	16.406	17.829	32.839	33.012	23.097	38.637	37.980	36.641
	900	Min	**44.195**	46.969	45.842	47.344	107.407	106.832	67.939	132.910	131.491	130.931
		Max	**44.310**	48.608	47.310	49.409	121.232	107.571	68.224	139.303	135.710	132.904
		Avg	**44.234**	47.815	46.140	48.094	115.012	107.174	68.018	135.926	133.278	131.848
	1500	Min	**73.733**	77.107	75.205	76.916	193.523	189.848	113.247	248.315	245.764	248.212
		Max	**75.611**	78.529	77.193	78.873	201.949	192.243	113.367	260.620	252.230	253.200
		Avg	**74.095**	77.772	76.394	78.122	197.262	191.025	113.305	253.714	249.900	250.838
D_9	300	Min	**20.098**	23.697	22.668	23.842	38.213	36.604	37.460	43.877	43.646	42.599
		Max	**20.174**	24.954	23.046	24.824	38.548	36.794	37.628	45.133	47.313	43.088
		Avg	**20.126**	24.223	22.818	24.297	38.421	36.713	37.548	44.439	44.296	42.790
	900	Min	**60.023**	63.854	62.451	64.003	120.953	118.471	89.919	145.600	146.591	141.441
		Max	**60.302**	66.557	64.231	67.714	121.867	119.243	97.388	149.600	170.342	161.315
		Avg	**60.181**	64.803	62.795	64.750	121.292	118.856	92.890	146.600	152.863	150.354
	1500	Min	**99.961**	104.231	102.648	104.687	214.429	213.021	177.504	278.953	268.821	267.217
		Max	**100.364**	106.827	105.310	107.602	217.676	244.700	203.813	309.324	323.782	283.917
		Avg	**100.160**	105.578	104.546	106.389	215.562	224.760	190.212	291.425	286.531	274.623

HABC, Heuristic Task Scheduling with Artificial Bee Colony; ACO, Ant Colony Optimization; PSO, Particle Swarm Optimization; IPSO, improved PSO; FCFS, First Come First Serve; SJF, Smallest Job First; LJF, Largest Job First.

Table 9 The comparison of the average makespan of the systems with HABC, ACO, PSO, and IPSO in a heterogeneous environment with datasets D_7 – D_9.

Dataset	Task	Variable	HABC_LJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_FCFS	IPSO_LJF	IPSO_SJF
D_7	300	Min	**7.157**	7.739	7.289	8.396	24.196	22.050	24.566	26.869	26.160	26.898
		Max	**7.173**	8.962	7.622	9.255	27.089	22.625	24.966	35.178	26.884	27.933
		Avg	**7.160**	8.500	7.444	8.802	25.044	22.279	24.726	29.741	26.446	27.167
	900	Min	**19.203**	20.137	19.238	20.553	78.132	72.733	76.368	98.286	97.163	98.379
		Max	**19.441**	21.740	19.828	21.508	107.029	73.530	77.370	104.773	99.165	100.482
		Avg	**19.333**	20.989	19.542	20.926	86.726	73.098	76.781	100.391	98.151	99.511
	1500	Min	**31.908**	32.734	31.960	33.023	134.903	137.902	138.639	202.683	199.200	200.864
		Max	**32.326**	34.789	32.494	35.537	150.393	140.233	141.081	214.879	204.439	206.623
		Avg	**32.045**	33.630	32.307	33.614	139.391	139.147	139.769	206.454	201.669	203.649
D_8	300	Min	**12.967**	14.688	13.103	15.059	29.644	27.963	31.369	32.367	31.856	33.968
		Max	**13.041**	16.495	14.136	16.599	32.058	28.574	31.682	34.567	32.874	37.191
		Avg	**12.994**	15.451	13.617	15.689	30.879	28.262	31.464	33.658	32.236	34.868
	900	Min	**36.784**	37.921	37.188	38.609	90.470	88.877	95.021	113.357	111.705	113.662
		Max	**37.245**	41.936	37.716	40.397	93.650	91.074	113.524	117.442	113.933	116.047
		Avg	**36.984**	39.784	37.400	39.208	91.836	89.756	102.922	115.047	112.727	114.897
	1500	Min	**60.618**	61.338	60.874	62.194	163.189	162.642	165.601	225.678	223.370	225.126
		Max	**60.920**	66.208	61.390	63.100	167.998	165.742	177.877	235.020	228.155	230.726
		Avg	**60.728**	63.599	61.113	62.608	166.105	163.988	171.501	229.573	225.446	227.433
D_9	300	Min	**18.040**	20.069	18.699	20.497	35.894	32.684	36.773	38.833	36.874	39.586
		Max	**18.094**	21.962	19.877	21.893	37.445	33.098	37.211	40.411	37.782	40.017
		Avg	**18.051**	20.827	19.253	21.114	36.423	32.913	36.958	39.699	37.208	39.750
	900	Min	50.580	**50.090**	50.979	52.478	103.374	99.892	103.903	126.204	124.505	127.557
		Max	**50.879**	54.592	51.889	53.356	104.394	101.280	104.900	131.245	127.389	130.504
		Avg	**50.670**	52.765	51.231	52.859	103.951	100.736	104.504	128.347	125.549	128.516
	1500	Min	**82.976**	83.644	83.184	84.285	182.045	180.425	182.410	243.361	241.335	243.867
		Max	84.526	90.443	**83.923**	85.677	187.234	184.087	184.573	255.820	246.441	249.162
		Avg	**83.412**	85.631	83.573	84.847	183.959	181.961	183.547	247.869	243.910	245.499

HABC, Heuristic Task Scheduling with Artificial Bee Colony; ACO, Ant Colony Optimization; PSO, Particle Swarm Optimization; IPSO, improved PSO; FCFS, First Come First Serve; SJF, Smallest Job First; LJF, Largest Job First.

Table 10 The comparison of the average makespan of the systems with HABC, ACO, PSO, and IPSO in a homogeneous environment with datasets D_{10} – D_{12}.

Dataset	Task	Variable	HABC_LJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_FCFS	IPSO_LJF	IPSO_SJF
D_{10}	300	Min	**9.840**	11.152	10.554	11.320	31.987	27.146	28.775	29.970	29.706	30.153
		Max	**9.943**	12.283	10.891	12.288	36.282	28.566	29.783	34.790	31.869	32.929
		Avg	**9.857**	11.583	10.739	11.773	34.396	28.004	29.273	31.175	30.264	31.182
	900	Min	**29.170**	30.798	29.866	30.870	96.423	89.398	91.702	110.496	108.828	109.847
		Max	**29.390**	31.637	30.393	31.946	101.850	97.457	107.651	114.164	112.227	125.505
		Avg	**29.284**	31.287	30.171	31.318	98.688	93.432	99.153	111.847	110.056	114.298
	1500	Min	**48.753**	50.840	49.349	51.067	168.387	170.806	171.145	236.442	216.480	217.155
		Max	**49.059**	51.677	50.318	51.767	175.637	180.935	180.160	261.561	245.113	256.378
		Avg	**48.904**	51.238	49.811	51.375	172.856	175.650	175.458	246.532	225.088	224.397
D_{11}	300	Min	**17.111**	19.555	18.880	20.020	37.320	34.579	34.573	38.583	37.057	37.894
		Max	**17.142**	20.752	19.158	21.054	38.739	40.478	35.841	40.244	41.899	38.477
		Avg	**17.124**	20.208	19.026	20.398	38.062	36.998	35.571	39.567	39.499	38.056
	900	Min	**51.537**	54.776	53.445	54.654	115.854	109.765	111.203	131.665	129.210	131.846
		Max	**51.832**	56.250	54.977	57.353	117.708	110.733	112.638	149.719	146.301	153.488
		Avg	**51.637**	55.537	53.830	55.320	116.586	110.284	111.802	136.028	134.417	138.309
	1500	Min	**85.919**	89.375	88.298	89.697	215.555	196.344	199.781	262.457	261.945	261.034
		Max	**86.397**	90.958	89.611	91.721	238.502	201.305	200.884	324.247	278.191	284.655
		Avg	**86.059**	90.182	89.136	90.462	224.720	199.123	200.360	332.678	265.998	269.802
D_{12}	300	Min	**22.244**	25.945	24.950	26.195	40.482	40.616	39.496	46.954	45.572	44.392
		Max	**22.371**	27.360	25.323	27.377	40.875	43.034	40.046	56.344	50.751	50.231
		Avg	**22.274**	26.432	25.186	26.673	40.662	42.176	39.677	49.659	47.707	46.474
	900	Min	**67.354**	70.989	70.207	71.414	129.654	127.078	126.089	149.600	149.927	153.897
		Max	**67.617**	73.371	72.689	75.204	130.476	135.066	128.023	178.262	182.307	171.051
		Avg	**67.514**	72.282	70.634	72.271	130.069	131.126	126.724	159.679	158.755	159.720
	1500	Min	**112.013**	117.006	115.147	116.653	239.050	222.857	223.367	287.868	285.495	286.239
		Max	**112.415**	118.822	117.796	119.924	256.957	227.268	226.719	298.553	290.658	291.120
		Avg	**112.265**	117.840	117.247	118.586	246.514	224.318	225.411	292.954	288.298	288.380

HABC, Heuristic Task Scheduling with Artificial Bee Colony; ACO, Ant Colony Optimization; PSO, Particle Swarm Optimization; IPSO, improved PSO; FCFS, First Come First Serve; SJF, Smallest Job First; LJF, Largest Job First.

Table 11 The comparison of the average makespan of the systems with HABC, ACO, PSO, and IPSO in a heterogeneous environment with datasets D_{10} – D_{12}.

Dataset	Task	Variable	HABC_LJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_FCFS	IPSO_LJF	IPSO_SJF
D_{10}	300	Min	**8.570**	9.211	8.997	9.416	25.328	23.257	25.216	34.267	33.909	34.000
		Max	**8.594**	10.540	9.871	10.670	26.559	24.303	25.566	37.286	35.337	38.069
		Avg	**8.579**	9.945	9.332	10.127	25.791	23.569	25.368	35.458	34.199	36.175
	900	Min	**23.705**	24.778	24.090	25.086	79.953	77.752	79.527	134.436	122.920	106.875
		Max	**23.857**	26.676	24.448	26.233	80.804	79.033	80.539	149.669	136.827	132.263
		Avg	**23.765**	25.765	24.271	25.683	80.418	78.358	80.107	137.860	132.622	121.142
	1500	Min	**39.475**	40.010	39.631	40.767	145.304	143.807	145.584	208.322	204.563	207.098
		Max	**40.476**	42.386	40.197	43.376	148.738	146.613	150.235	217.963	210.574	212.015
		Avg	**39.875**	41.147	39.977	41.198	146.693	145.407	147.045	211.902	207.255	209.350
D_{11}	300	Min	**15.339**	16.487	15.730	16.377	31.885	29.381	32.504	37.556	36.849	37.450
		Max	**15.431**	18.755	16.581	18.339	34.454	29.743	33.134	47.160	38.005	40.015
		Avg	**15.369**	17.386	16.103	17.536	32.801	29.597	32.665	39.769	37.318	38.043
	900	Min	**42.742**	43.518	43.238	44.708	96.971	94.199	97.169	130.707	130.652	131.525
		Max	43.858	47.344	**43.846**	46.198	98.603	95.061	97.939	137.456	135.611	135.364
		Avg	**42.869**	45.593	43.540	45.321	97.502	94.616	97.492	133.879	131.903	133.123
	1500	Min	**70.269**	71.839	70.801	72.098	172.093	170.612	171.553	240.058	295.388	304.815
		Max	**70.547**	75.895	72.058	73.386	175.863	172.934	173.107	319.583	314.065	314.763
		Avg	**70.383**	73.419	71.294	72.809	173.570	171.435	172.327	303.582	306.437	308.351
D_{12}	300	Min	**20.479**	22.064	20.784	22.039	37.282	34.274	38.110	39.859	36.214	40.114
		Max	**20.554**	24.029	21.434	23.901	39.306	34.794	38.501	42.938	39.384	46.617
		Avg	**20.504**	22.897	21.048	22.927	37.732	34.544	38.310	40.846	37.953	42.435
	900	Min	**56.622**	56.887	56.785	57.182	108.605	105.402	109.153	129.440	126.990	130.697
		Max	**56.770**	63.778	57.721	60.232	110.586	106.888	110.214	135.108	129.848	132.790
		Avg	**56.673**	59.780	57.240	58.976	109.669	106.044	109.677	131.548	128.587	131.451
	1500	Min	**92.805**	94.120	93.016	94.593	190.609	189.282	190.582	250.071	246.614	340.863
		Max	**93.225**	98.253	93.849	95.619	194.868	192.231	192.333	263.459	350.374	356.086
		Avg	**92.951**	96.197	93.379	95.041	192.957	190.811	191.589	256.314	297.956	347.719

HABC, Heuristic Task Scheduling with Artificial Bee Colony; ACO, Ant Colony Optimization; PSO, Particle Swarm Optimization; IPSO, improved PSO; FCFS, First Come First Serve; SJF, Smallest Job First; LJF, Largest Job First.

Figure 2 Degrees of load imbalance in a homogeneous environment when Heuristic Task Scheduling with Artificial Bee Colony (HABC), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and improved PSO (IPSO) were used with (a) datasets D_1 - Random dataset, (b) datasets D_4 - Normal distribution dataset, (c) datasets D_7 - Random dataset with left-skewed distribution, (d) datasets D_{10} - Random dataset with right-skewed distribution.

the HABC algorithm (HABC_FCFS, HABC_LJF, HABC_SJF) performed lower average degrees of load imbalance than any of the compared algorithms (PSO and ACO algorithms) in the homogeneous environment.

In the same direction, the results in Figure 3 showed that the HABC algorithm (HABC_FCFS, HABC_LJF, HABC_SJF) outperformed the other two algorithms in the heterogeneous environment.

In the third part, the performance of the HABC algorithm in terms of Standard Deviation (S.D.) was evaluated. In order to ensure that the tasks are equally distributed into VMs and the system is balanced workload.

We evaluated the S.D. with dataset D_1 in Table 3. The number of groups of submitted tasks was varied in each run from 100 to 1500 submitted tasks in an increment of 100 tasks. The S.D. for 300, 600, 900, 1200, and 1500 tasks in cloud computing within homogeneous and heterogeneous environments are shown in Tables 12 and 13, respectively. Table 12 shows the S.D. of all tested algorithms with datasets (D_1) in homogeneous environment. The results show that HABC algorithm has less S.D. values than other methods. In the same direction of heterogeneous environment in Table 13.

The result of the experiment indicated that the makespan and the degree of imbalance are slight difference. On the other hand, if considered the performance measurement using S.D. between the HABC algorithm and ACO algorithm are very different. The results show that HABC algorithm highly enhanced the S.D. compared to the other algorithms in cloud computing within homogeneous and heterogeneous environments. The goal is to achieve a balance between the resources in the system and to minimize the makespan time.

To summarize, even though all of the presented algorithms utilize a similar scheduling process based on a heuristic method. When they were compared, the HABC algorithm was able to show a better performance than those shown by ACO, PSO, and IPSO algorithms, especially in the case of using large job submitting first

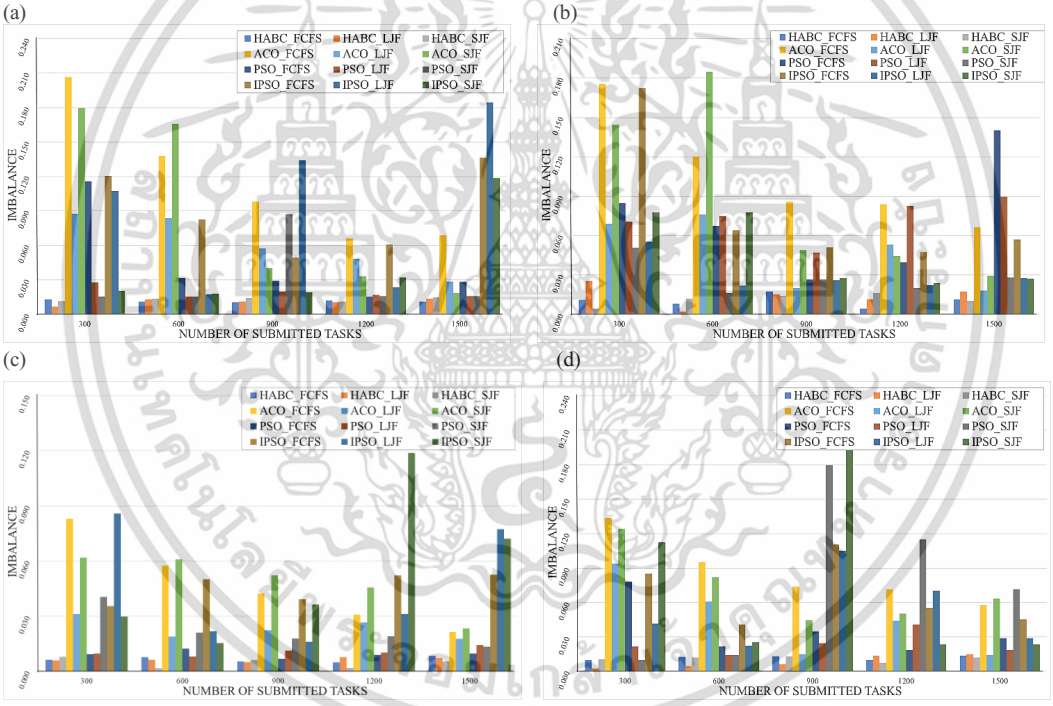


Figure 3 | Degrees of load imbalance in a heterogeneous environment when Heuristic Task Scheduling with Artificial Bee Colony (HABC), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and improved PSO (IPSO) were used with (a) datasets D_1 - Random dataset, (b) datasets D_4 - Normal distribution dataset, (c) datasets D_7 - Random dataset with left-skewed distribution, (d) datasets D_{10} - Random dataset with right-skewed distribution.

Table 12 | The comparison of Standard Deviation with HABC, ACO, PSO, and IPSO in a homogeneous environment with datasets D_1.

Number of Tasks	HABC_ FCFS	HABC_ LJF	HABC_ SJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_ FCFS	IPSO_LJF	IPSO_SJF
300	**0.12**	**0.06**	**0.09**	1.09	1.01	1.12	0.51	0.38	0.46	0.21	0.14	0.16
600	**0.11**	**0.05**	**0.08**	2.07	2.03	2.09	0.51	0.38	0.45	0.21	0.14	0.15
900	**0.09**	**0.04**	**0.07**	3.02	2.99	3.04	0.48	0.37	0.44	0.18	0.13	0.14
1200	**0.08**	**0.04**	**0.08**	4.08	4.06	4.1	0.49	0.37	0.45	0.19	0.13	0.15
1500	**0.09**	**0.07**	**0.09**	5.05	5.03	5.06	0.52	0.36	0.44	0.22	0.12	0.14

HABC, Heuristic Task Scheduling with Artificial Bee Colony; ACO, Ant Colony Optimization; PSO, Particle Swarm Optimization; IPSO, improved PSO; FCFS, First Come First Serve; SJF, Smallest Job First; LJF, Largest Job First.

Table 13 The comparison of standard deviation with HABC, ACO, PSO, and IPSO in a heterogeneous environment with datasets D_1.

Number of Tasks	HABC_ FCFS	HABC_ LJF	HABC_ SJF	ACO_FCFS	ACO_LJF	ACO_SJF	PSO_FCFS	PSO_LJF	PSO_SJF	IPSO_ FCFS	IPSO_LJF	IPSO_SJF
300	**0.16**	**0.12**	**0.14**	0.71	0.63	0.74	0.86	0.52	1.18	0.41	0.22	0.48
600	**0.17**	**0.09**	**0.15**	1.32	1.27	1.33	0.89	0.55	1.29	0.44	0.25	0.59
900	**0.15**	**0.12**	**0.13**	1.93	1.89	1.9	0.94	0.52	1.26	0.49	0.22	0.56
1200	**0.17**	**0.13**	**0.12**	2.57	2.56	2.59	0.87	0.53	1.03	0.42	0.23	0.33
1500	**0.16**	**0.11**	**0.13**	3.19	3.17	3.2	0.82	0.65	0.73	0.37	0.25	0.23

HABC, Heuristic Task Scheduling with Artificial Bee Colony; ACO, Ant Colony Optimization; PSO, Particle Swarm Optimization; IPSO, improved PSO; FCFS, First Come First Serve; SJF, Smallest Job First; LJF, Largest Job First.

(HABC_LJF). Moreover, the system still balanced the load of submitting tasks adequately with minimum makespan.

5. CONCLUSION

In this paper, to improve task scheduling and load balancing, an algorithm for VMs in cloud computing based on heuristic task scheduling called HABC for VMs in cloud computing within both homogeneous and heterogeneous environments is proposed. The HABC was implemented on a cloud computing model in order to optimize the process of task scheduling and load balancing of volumes of data. A detailed comparison between the performances of the proposed method and other compared algorithms including ACO algorithms, PSO algorithms, and improved PSO algorithms is also presented.

The experiments were conducted with four types of datasets in order to measure the performance of our algorithm in terms of minimum makespan, maximum makespan, average makespan, and degree of load imbalance. The results show that the HABC with Largest Job First heuristic algorithm (HABC_LJF) offers the best performance in scheduling and load balancing.

This novel-proposed approach can serve as a useful alternative for cloud computing within homogeneous and heterogeneous environments. However, this work was a simulation on a few datasets. In future works, we will further conduct similar experiments on larger datasets. We plan to apply this HABC method in other practical applications to real-world datasets and will explore the possibility of HABC to handle multiple job scheduling with different priorities.

COMPLIANCE WITH ETHICAL STANDARDS

CONFLICT OF INTEREST

The author declares that he has no conflict of interest against any company or institution.

ETHICAL STANDARDS

This research has not involved human participants and/or animals, except for the author contribution.

ACKNOWLEDGMENTS

Our appreciation goes to the Department of Computer Science, Faculty of Science, King Mongkut's Institute of Technology Ladkrabang for supporting the facilities in doing this research.

REFERENCES

[1] D. Karaboga, An Idea Based on Honey Bee Swarm for Numerical Optimization, Technical Report-tr06, Erciyes University, 2005. https://pdfs.semanticscholar.org/015d/f4d97ed1f541752842c49d12e429a785460b.pdf

[2] B. Akay, D. Karaboga, A modified artificial bee colony algorithm for real-parameter optimization, Inf. Sci. 192 (2012), 120–142.

[3] T. Davidović, D. Teodorović, M. Selmic, Bee colony optimization-part I: the algorithm overview, Yugoslav J. Oper. Res. 25 (2015), 33–56.

[4] K. Benatchba, L. Admane, M. Koudil, Using bees to solve a data-mining problem expressed as a max-sat one, in: J. Mira, J.R. Álvarez (Eds.), Artificial intelligence and knowledge engineering applications: a bioinspired approach, IWINAC 2005. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 3562, (2005), pp. 212–220.

[5] M.A.M. Shukran, Y.Y. Chung, W.C. Yeh, N. Wahid, A.M.A. Zaidi, Artificial bee colony-based data mining algorithms for classification tasks, Mod. Appl. Sci. 5 (2011), 217–231.

[6] L.P. Wong, C.Y. Puan, M.Y.H. Low, C.S. Chong, Bee colony optimization algorithm with big valley landscape exploitation for job shop scheduling problems, in Proceedings of the 2008 Winter Simulation Conference, Miami, FL, USA, 2008, pp. 2050–2058.

[7] R. Zhang, S. Song, C. Wu, A hybrid artificial bee colony algorithm for the job shop scheduling problem, Int. J. Prod. Econ. 141 (2013), 167–178.

[8] M.H. Kashan, N. Nahavandi, A.H. Kashan, DisABC: a new artificial bee colony algorithm for binary optimization, Appl. Soft. Comput. 12 (2012), 342–352.

[9] G. Pampara, A.P. Engelbercht, Binary artificial bee colony optimization, in Proceedings of the 2011 IEEE Symposium on Swarm Intelligence, Paris, France, 2011, pp. 1–8.

[10] M.S. Kiran, The continuous artificial bee colony algorithm for binary optimization, Appl. Soft. Comput. 33 (2015), 15–23.

[11] N. Pathak, S.P. Tiwari, Travelling salesman problem using bee colony with spv, Int. J. Soft Comput. Eng. 2 (2012), 410–414. http://www.ijsce.org/wp-content/uploads/papers/v2i3/C0716052312.pdf

[12] W.C. Yen, T.J. Hsieh, Artificial bee colony algorithm-neural networks for S-system models of biochemical networks approximation, Neural. Comput. Appl. 21 (2012), 365–375.
[13] C. Dong, Z. Xiong, X. Liu, Y. Ye, Y. Yang, W. Guo, Dual-search artificial bee colony algorithm for engineering Optimization, IEEE Access. 7 (2019), 24571–24584.
[14] S. Aslan, A Transition control mechanism for Artificial Bee Colony (ABC) algorithm, Comput. Intel. Neurosc. 2019 (2019), 1–24.
[15] A. Draa, A. Bouaziz, An artificial bee colony algorithm for image contrast enhancement, Swarm. Evol. Comput. 16 (2014), 69–84.
[16] H. Li, W. Li, Enhanced artificial bee colony algorithm and its application in multi-threshold image feature retrieval, Multimed. Tools. Appl. 78 (2019), 8683–8698.
[17] C. Ozturk, E. Hancer, D. Karaboga, Improved clustering criterion for image clustering with artificial bee colony algorithm, Pattern. Anal. Appl. 18 (2015), 587–599.
[18] T. Mizan, S.M.R.A. Masud, R. Latip, Modified bees life algorithm for job scheduling in hybrid cloud, Int. J. Eng. Technol. 2 (2012), 974–979. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=934F121C8F2C8CB09212BA0478C4E4A5?doi=10.1.1.412.902&rep=rep1&type=pdf
[19] B. Kruekaew, W. Kimpan, Virtual machine scheduling management on cloud computing using artificial bee colony, in Proceedings of the International MultiConference of Engineers and Computer Scientists, Hong Kong, China, 2014, pp. 18–22. https://pdfs.semanticscholar.org/b671/240dc9bc3a05b06338068641dcf21b999879.pdf
[20] W. Kimpan, B. Kruekaew, Heuristic task scheduling with artificial bee colony algorithm for virtual machines, in Proceedings of the 2016 Joint 8th International Conference on Soft Computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS), IEEE, Sapporo, Japan, 2016, pp. 281–286.
[21] Poonam, M. Dutta, N. Aggarwal, Meta-heuristics based approach for workflow scheduling in cloud computing: a survey, in: S.S. Dash, M. Bhaskar, B.K. Panigrahi, S. Das (Eds.), Proceedings of Artificial Intelligence and Evolutionary Computations in Engineering Systems, Advances in Intelligent Systems and Computing, Springer, New Delhi, 394, (2016), pp. 1331–1345.
[22] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, D.A. Patterson, A. Rabkin, I. Stoica, M. Zaharua, Above the Clouds: a Berkeley View of Cloud Computing, Report UCB/EECS, Department of Electrical Engineering and Computer Sciences, University of California, Berkley, CA, USA, 2009. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf
[23] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility, Future. Gener. Comp. Syst. 25 (2009), 599–616.
[24] M. Armbrust, A. Fox, R. Griffith, A view of cloud computing, Commun. ACM. 53 (2010), 50–58.
[25] S. Chaisiri, B.S. Lee, D. Niyato, Optimization of resource provisioning cost in cloud computing, IEEE. T. Serv. Comput. 5 (2011), 164–177.
[26] F. Xu, F. Liu, H. Jin, A.V. Vasilakos, Managing performance overhead of virtual machines in cloud computing: a survey, state of the art, and future directions, Proc. IEEE. 102 (2013), 11–31.
[27] P.D. Bharathi, P. Prakash, M.V.M. Kiran, Virtual machine placement strategies in cloud computing, in Proceedings of the 2017 Innovations in Power and Advanced Computing Technologies (i-PACT), IEEE, Vellore, India, 2017, pp. 1–7.
[28] M.R. Garey, D.S. Johnson, Computers and Intractability: a Guide to the Theory of NP-Completeness, WH Freeman, San Francisco, 1979, p. 338.
[29] J.D. Ullman, NP-complete scheduling problems, J. Comput. Syst. Sci. 10 (1975), 384–393.
[30] Q. Zhang, L. Cheng, R. Boutaba, Cloud computing: state-of-the-art and research challenges, J. Int. Serv. Appl. 1 (2010), 7–18.
[31] J.W.M. Bush, B.A. Thurber, F. Blanchette, Particle clouds in homogeneous and stratified environments, J. Fluid Mech. 489 (2003), 29–54.
[32] S. Crago, K. Dunn, P. Eads, L. Hochstein, D.I. Kang, Heterogeneous cloud computing, in Proceedings of the 2011 IEEE International Conference on Cluster Computing, IEEE, Austin, TX, USA, 2011, pp. 378–385.
[33] J.S. Chase, D.C. Anderson, P.N. Thakar, A.M. Vahdat, R.P. Doyle, Managing energy and server resources in hosting centres, ACM SIGOPS Oper. Syst. Rev. 35 (2001), 103–116.
[34] D. Kusic, J.O. Kephart, J.E. Hanson, N. Kandasamy, G. Jiang, Power and performance management of virtualized computing environments via lookahead control, Cluster. Comput. 12 (2009), 1–15.
[35] D. Minarolli, B. Freisleben, Utility-based resource allocation for virtual machines in cloud computing, in Proceedings of the 2011 IEEE Symposium on Computers and Communications (ISCC), IEEE, Kerkyra, Greece, 2011, pp. 410–417.
[36] A. Gorbenko, V. Popov, Task-resource scheduling problem, Int. J. Automat. Comput. 9 (2012), 429–441.
[37] S. Son, G. Jung, S. Chan, An SLA-based cloud computing that facilitates resource allocation in the distributed data centers of a cloud provider, J. Supercomput. 64 (2013), 606–637.
[38] A. Singh, K. Dutta, Apply AHP for resource allocation problem in cloud, J. Comput. Commun. 3 (2015), 13–21.
[39] Q.Z. Ullah, S. Hassan, G.M. Khan, Adaptive resource utilization prediction system for infrastructure as a service cloud, Comput. Intel. Neurosc. 2017 (2017), 1–12.
[40] H. Rastegarfar, L.A. Rusch, A. Leon-Garcia, Optical load-balancing tradeoffs in wavelength-routing cloud data centers, J. Optical Commun. Netw. 7 (2015), 286–300.
[41] F. Tang, L.T. Tang, C. Tang, J. Li, M. Guo, A dynamical and load-balanced flow scheduling approach for big data centers in clouds, IEEE Trans. Cloud Comput. 6 (2016), 915–828.
[42] J. Wan, B. Chen, S. Wang, M. Xia, D. Li, C. Liu, Fog computing for energy-aware load balancing and scheduling in smart factory, IEEE. Trans. Ind. Inform. 14 (2018), 4548–4556.
[43] R. Bagheri, M. Jahanshahi, Scheduling workflow applications on the heterogeneous cloud resources, Indian J. Sci. Technol. 8 (2015), 1–8.
[44] E.M. Mocanu, M. Florea, M.I. Andreica, N. Tapus, Cloud computing-task scheduling based on genetic algorithms, in Proceedings of the 2012 IEEE International Systems Confernce SysCon, IEEE, Vancouver, BC, Canada, 2012, pp. 1–6.
[45] K. Dasgupta, B. Mandal, P. Dutta, J.K. Mnda, S. Dam, A Genetic Algorithm (GA) based load balancing strategy for cloud computing, Proc. Technol. 10 (2013), 340–347.

[46] S. Pandey, L. Wu, S.M. Guru, R. Buyya, A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments, in Proceedings of the 2010 24th IEEE International Conference on Advance Information Networking and Applications, IEEE, Perth, Australia, 2010, pp. 400–407.
[47] K. Zhu, H. Song, L. Liu, J. Gao, G. Cheng, Hybrid genetic algorithm for cloud computing applications, in Proceedings of the 2011 IEEE Asia-Pacific Services Computing Conference (APSCC), IEEE, Jeju Island, South Korea, 2011, pp. 182–187.
[48] H.S. Al-Olima, M. Alam, R. Green, J.K. Lee, Cloudlet scheduling with particle swarm optimization, in Proceedings of the 2015 Fifth International Conference on Communication Systems and Network Technologies, IEEE, Gwalior, India, 2015, pp. 991–995.
[49] H. Saleh, H. Nashaat, W. Saber, H.M. Harb, IPSO task scheduling algorithm for large scale data in cloud computing environment, IEEE Access. 7 (2018), 5412–5420.
[50] N.J. Navimipour, F.S. Milani, Task scheduling in the cloud computing based on the cuckoo search algorithm, Int. J. Model. Optim. 5 (2015), 44–47.
[51] S. Aujla, A. Ummat, Task scheduling in cloud using hybrid cuckoo algorithm, Int. J. Comput. Netw. Appl. 2 (2015), 144–150. https://www.ijcna.org/Manuscripts/Volume-2/Issue-3/Vol-2-issue-3-M-04.pdf
[52] M. Agarwal, G.M.S. Srivastava, A Cuckoo search algorithm-based task scheduling in cloud computing, in: S.K. Bhatia, K.K. Mishra, S. Tiwari, V.K. Singh (Eds.), Proceedings of Advances in Computer and Computational Science, Advances in Intelligent Systems and Computing, Springer, Singapore, 554 (2018), pp. 293–299.
[53] X.S. Yang, S. Deb, Cuckoo search: recent advances and applications, Neural Comput. Appl. 24 (2014), 169–174.
[54] T.P. Jacob, K. Pradeep, A multi objective optimal task scheduling in cloud environment using cuckoo particle swarm optimization, Wireless Pers. Commun. 109 (2019), 315–331.
[55] Y. Zhou, X. Huang, Scheduling workflow in cloud computing based on ant colony optimization algorithm, in Proceedings of the 2013 6th International Conference on Business Intelligence and Financial Engineering, IEEE, Hangzhou, China, 2013, pp. 57–61.
[56] K. Li, G. Xu, G. Zhao, Y. Dong, D. Wang, Cloud task scheduling based on load balancing ant colony optimization, in Proceedings of the 2011 6th Annual China Grid Conference, IEEE, Liaoning, China, 2011, pp. 3–9.
[57] W. Li, H. Yan-xiang, Web service composition based on QoS with chaos particle swarm optimization, in Proceedings of the 2010 6th International Conference on Wireless Communications Networking and Mobile Computing, IEEE, Chengdu, China, 2010, pp. 1–4.
[58] L. Wang, J. Shen, J. Luo, F. Dong, An improved genetic algorithm for cost-effective data-intensive service composition, in Proceedings of the 2013 Ninth International Conference on Semantics, Knowledge and Grids, IEEE, Beijing, China, 2014, pp. 105–112.
[59] L. Wang, J. Shen, Multi-phase ant colony system for multi-party data-intensive service provision, IEEE Trans. Serv. Comput. 9 (2016), 264–276.
[60] M.H. Ferdaus, M. Murshed, R.N. Calheiros, R. Buyya, Multi-objective, decentralized dynamic virtual machine consolidation using ACO Metaheuristic in Computing Clouds, in Concurrency Computation Practice Experience, Wiley InterScience, 2016, pp. 1–40. arXiv:1706.06646. https://arxiv.org/abs/1706.06646
[61] S.K. Gavvala, C. Jatoth, G. Gangadharan, R. Buyya, Qos-aware cloud service composition using eagle strategy, Future Gener. Comput. Syst. 90 (2019), 273–290.
[62] D. Karaboga, B. Akay, A survey: algorithms simulating bee swarm intelligence, Artif. Intell. Rev. 31 (2009), 68–85.
[63] D. Ji, The application of artificial bee colony (ABC) algorithm in FIR filter design. in Proceedings of the 2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), IEEE, Changsha, China, 2016, pp. 663–667.
[64] A.J. Hosios, J.M. Rousseau, A Heuristic scheduling algorithm, J. Oper. Res. Soc. 31 (1980), 749–753.
[65] Y. Dai, X. Zhang, A synthesized heuristic task scheduling algorithm, Sci. World. J. 2014 (2014), 1–9.
[66] Z.R.M. Azmi, K.A. Bakar, M.S. Shamsir, W.N.W. Manan, A.H. Abdullah, Performance comparison of priority rule scheduling algorithms using different inter arrival time jobs in grid environment. Int. J. Grid Distrib. Comput. 4 (2011), 61–70. http://article.nadiapub.com/IJGDC/vol4_no3/5.pdf
[67] M.D. George, W. Brown, Standard deviation, standard error: which 'standard' should we use?, Am. J. Dis. Child. 136 (1982), 937–941.
[68] R.N. Calheiros, R. Ranjan, C.A.F.D. Rose, R. Buyya, CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. Pract. Exper. 41 (2010), 23–50.
[69] R.N. Calheiros, R. Ranjan, C.A.F.D. Rose, R. Buyya, CloudSim: a Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services, Technical Report, GRIDS-TR-2009-1, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Melbourne, Australia, 2009. arXiv:0903.2525. https://arxiv.org/abs/0903.2525
[70] A. Azzalini, A.D. Valle, The multivariate skew-normal distribution, Biometrika. 83 (1996), 715–726.

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 4, 2021, accepted February 4, 2022, date of publication February 9, 2022, date of current version February 17, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3149955

Multi-Objective Task Scheduling Optimization for Load Balancing in Cloud Computing Environment Using Hybrid Artificial Bee Colony Algorithm With Reinforcement Learning

BOONHATAI KRUEKAEW AND WARANGKHANA KIMPAN, (Member, IEEE)
Department of Computer Science, School of Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand
Corresponding author: Warangkhana Kimpan (warangkhana.ki@kmitl.ac.th)

This work was supported by the School of Science, King Mongkut's Institute Technology Ladkrabang.

ABSTRACT Workload balancing in cloud computing is still challenging problem, especially in Infrastructure as a Service (IaaS) in the cloud model. A problem that should not occur during cloud access is a host or server being overloaded or underloaded, which may affect the processing time or may result in a system crash. Therefore, to prevent these problems, an appropriate schedule of access should be considered so that the system can distribute tasks across all available resources, which is called load balancing. The load balancing technique should ensure that all Virtual Machines (VMs) are used appropriately. In this paper, an independent task scheduling approach in cloud computing is proposed using a Multi-objective task scheduling optimization based on the Artificial Bee Colony Algorithm (ABC) with a Q-learning algorithm,which is a reinforcement learning technique that helps the ABC algorithm work faster, called the MOABCQ method. The proposed method aims to optimize scheduling and resource utilization, maximize VM throughput, and create load balancing between VMs based on makespan, cost, and resource utilization, which are limitations of concurrent considerations. Performance analysis of the proposed method was compared using CloudSim with the existing load balancing and scheduling algorithms: Max-Min, FCFS, HABC_LJF, Q-learning, MOPSO, and MOCS algorithms in three datasets: Random, Google Cloud Jobs (GoCJ), and Synthetic workload. The experimental results indicated that the algorithms used MOABCQ approach outperformed the other algorithms in terms of reducing makespan, reducing cost, reducing degree of imbalance, increasing throughput and average resource utilization.

INDEX TERMS Cloud computing, hybrid artificial bee colony algorithm, multi-objective task, Q-learning, task scheduling.

I. INTRODUCTION

Recently, cloud computing has played an important role in many organizations. Cloud computing started out to provide of users requirements for accessing resource computing or to enable users to purchase cloud services as required within the concept of on-demand resource sharing through highly internet-based applications. Cloud computing can also process many different services depending on the service and working platforms that are needed by the users [1]. Cloud computing is a combination of distributed and parallel computing with the sharing of resources such as software and hardware that will be utilized as ''pay-as-you-go'' [2]. To use it, users do not need to purchase any platform or software; they only have an internet connection to access and pay for their usage.

Infrastructure as a Service (IaaS) is one of the technology models behind the management of servers, data centers, and Virtual Machines (VMs). IaaS is a cloud service that provides a cloud computer or server for processing and storing data in the cloud. Users can run any operating system or application on a rental server free of maintenance and operating costs of hardware. Since the cloud infrastructure is run on VMs, IaaS has another advantage, including providing the access of the user to the servers in nearby locations. This service depends on the needs of the user, called Quality of Service (QoS) and Service Level Agreements (SLAs). The payment for the services depends on the agreement between the user and

The associate editor coordinating the review of this manuscript and approving it for publication was Gerard P. Parr.

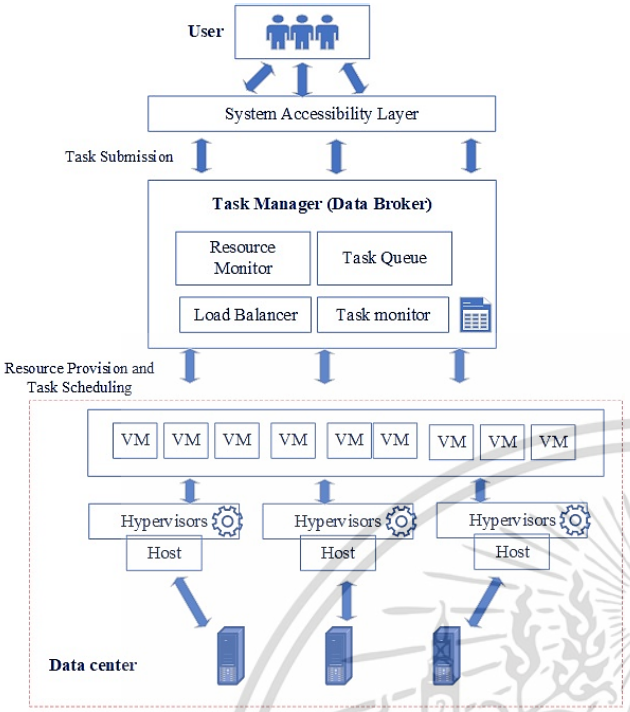


FIGURE 1. Cloud resource management framework.

the Cloud Service Providers (CSPs) [3]. Moreover, Cloud computing enables CSPs to provide data centers with high-performance computing resources to support users accessing cloud services.

A VM consumes the resources of the host machine: RAM, hard disk, or CPU. The resources on the VM are required by the requested resource for operation. As a result, the cloud network has an unequal resource distribution, and some VMs may not access the resources they require because many VMs have preemptive and non-preemptive connections to resources [4].

When a task is sent off to be processed in the cloud, the VM should be able to operate quickly to reduce waiting time. However, tasks should be distributed among all VMs in parallel to balance the system and ensure efficient use of available resources. For this reason, task scheduling to be assigned and distributed across existing resources is necessary. When multiple tasks are assigned to one VM or multiple VMs, the assigned tasks will run concurrently to complete the assignments. Therefore, the assignment must be ensured that not all tasks are loaded on only a single VM, which will cause other VMs to become inaccessible or imbalanced in the system. To avoid this, other conditions must be considered in scheduling, such as makespan, cost, and resource utilization. Several researchers have proposed ideas to manage load balancing in both heterogeneous [5] and homogeneous environments [6]. The main goal of allocating tasksof a system in a load balance state is to optimize the distribution of tasks across existing resources and to reduce the system processing time.

The cloud resource management framework is shown in Fig. 1. When a user submits a request into the system, the task is passed to the cloud broker, which is the main objective that researchers should focus on providing effective algorithms. The proposed method should efficiently submit tasks to the appropriate VM by the user depending on the required parameters, such as the deadline or other requirements. It must ensure that the user submitted the requests that are fulfilled in accordance with the requirements specified in the SLA document. Users make their requests over the internet, and the requests are stored in VMs. Then, the CSPs deliver QoS-based queries to ensure that the requests of the user can be processed as intended. This process depends on the performance of the scheduling policy (Data Broker). In this step, the workload balance must be adjusted between the machine and the server. Cloud computing depends highly on Virtual Machine Monitor (VMM) or hypervisors. Hypervisors help operate multiple VMs in the same hardware layer [7]. VMware is one of the most famous software services efficiently used for managing server resources in organizations. Even though virtualization plays an important role in cloud technology, problems still often arise, such as improper scheduling or handing over tasks to VMs that cannot meet the requests. To solve this problem, scheduling and load balancing between nodes in cloud computing should be implemented to optimize resource utilization. Scheduling and load balancing between nodes also resulted in a reduction in processing time and an imbalance in the system.

Task allocation to match the right resource, such as time, cost, and success rate, must be considered in cloud computing. Many studies have discussed scheduling or optimizing resources in the cloud by single or bi-objective optimization of QoS parameters [8]–[10], and several articles have discussed multi-objective optimization [11]. Examples of algorithms that were used to increase the efficiency of on-demand tasks such as Particle Swarm Optimization (PSO) [12], [13], Genetic Algorithm (GA) [14], niched Pareto genetic algorithm (NPGA) [15], and strength Pareto evolutionary algorithm (SPEA) [16].

The Artificial Bee Colony (ABC) algorithm is a meta-heuristic method used to solve problems and find solutions for an optimization answer that is close to the appropriate value developed by Karaboga [17], [18]. The ABC algorithm mimics the foraging behavior of bee colonies, which require adaptation to habitat and food environments. Several studies have demonstrated the effectiveness of the ABC algorithm in solving problems such as traveling salesman problem [19], and job shop scheduling problems [20]. The ABC algorithm works in a strategy similar to Reinforcement Learning (RL) with environment-based learning, which derives from observation to predict or decide a good solution. Agents learn how to behave in an environment by taking action and then observing the results for themselves. In this research, we use the Q-learning algorithm to help the system make predictions and decisions about choosing the appropriate schedule. Therefore, we propose a multi-objective optimization-scheduling

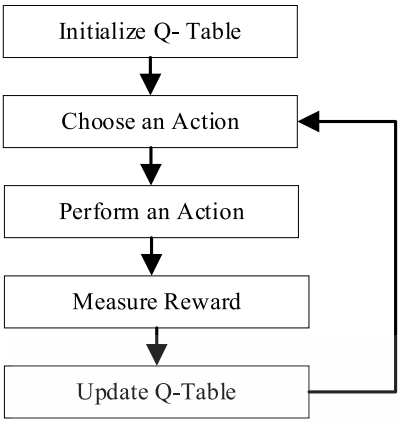


FIGURE 2. Q-learning algorithm process.

model using the ABC algorithm and Q-learning (MOABCQ) method for solving the scheduling problem in a cloud computing environment. The paper focuses on developing a multi-objective scheduling approach to optimize job scheduling with the goal of minimizing makespan, cost, and simultaneous use of resources, also considering the load balance of the system.

The main contributions of the proposed method can be described as follows:

- Formulate mathematical models for calculating resource utilization, makespan and cost for scheduling problems in cloud computing.
- Propose a multi-objective optimization scheduling model. This model consists of 3 objective functions defined as 1) execution time, 2) cost, and 3) utilization of resources. These are constraints in the proper task scheduling and the load balancing of the system.
- Propose a new approachthat is multi-objective optimization for the task scheduling problem using the Artificial Bee Colony algorithm (ABC) adapted with the implementation of the Q-learning algorithm, namely, MOABCQ, to optimize the multi-objective scheduling problem in cloud computing and to reduce the makespan, cost, and simultaneous use of resources. This new proposed method is combined with the First Come First Serve (FCFS) heuristic task scheduling called "MOABCQ_FCFS" and Largest Job First (LJF) heuristic task scheduling called "MOABCQ_LJF".
- Performance evaluation of our proposed method examined through CloudSim simulation. In this paper, we improved the classical Q-learning algorithm for load balancing in the cloud environment by combining it with ABC. This idea can improve the convergence rate, gain efficiency in load balancing, determine individual VM loads, and balance them through the fitness function.
- The proposed method was compared with other optimization algorithms, such as: First Come First Serve (FCFS) algorithm, Max-Min algorithm, Heuristic Task Scheduling with ABC with Largest Job First algorithm (HABC_LJF), Q-learning algorithm, multi-objective Particle Swarm Optimization (MOPSO) algorithm, and multi-objective Cuckoo Search (MOCS) algorithm. The three datasets used in this study are Random, Google Cloud Jobs (GoCJ), and Synthetic workload to observe in terms of makespan, cost, Average Resource Utilization Ratio (ARUR), throughput, and Degree of Imbalance (DI).

The rest of this article is structured as follows: Section II discusses the literature review. In Section III, the problem definition and formulation of the objective function will be described. The hybrid ABC algorithm for load balancing is presented in Section IV. In Section V, experimental evaluation and discussion will be explained, and finally, Section VI presents conclusions and future work.

II. LITERATURE REVIEW

Optimizing task scheduling and load balancing in cloud computing environments is known as an NP-hard problem [21]. Many studies have proposed optimization algorithms in cloud environment in various aspects, such as resource allocation, scheduling and load balancing procedures to reduce uptime, and system power consumption. In terms of efficient use of existing resources or fast processing, we need to consider an idea in scheduling multiple objectives or cloud environments.

Therefore, many ideas have been proposed using heuristic algorithms [22]–[29], meta-heuristic algorithm [30]–[37], hybrid meta-heuristic algorithm [14], [38], or even machine learning methods [39]–[41] to solve task scheduling and load balancing problems in the cloud computing environment. The related works are described as follows:

Many studies have used a heuristic algorithm to solve problems in a cloud computing environment. For example, A Max-Min based task scheduling algorithm was proposed by Mao *et al.* [22] to balance the load in the cloud and to predict the execution time of tasks. Then, the tasks were submitted to the VM based on their proposed Max-Min algorithm. The VM utilized more resources and decreased response time. Patel *et al.* [24] proposed an Enhanced Load balanced Min-Min (ELBMM) algorithm for static task scheduling. Jobs were assigned to VMs based on execution time, and the tasks were rescheduled to distribute to resources that idle.

Zhang *et al.* [25] studied the application of heuristic algorithms in cloud scheduling problems. They proposed a method for scheduling tasks to minimize the task completion time and execution cost (MCTE) in a smart grid-cloud system. Then, they performed mathematical modeling for the grid-cloud task scheduling problem.

Hussain *et al.* [26] proposed a resource-aware load-balancing algorithm (RALBA), which is a dynamic scheduling technique that maps independent and non-preemptive tasks to VMs. The process consisted of 2 parts. The first part was selecting the maximum size for the VM with the highest computational share. The second part was mapping the remaining tasks to the fastest working VM to perform. The results showed that RALBA was able to reduce makespan. However, there were some problems with

load imbalance and reducing resource utilization. The load balancing approach was proposed to be modified using soft computing-based stochastic hill climbing based on load balancing by Mondal *et al.* [27].

A dynamic multi-workflow scheduling algorithm named the competent dynamic multi-workflow scheduling (CDMWS) algorithm was proposed by Adhikari and Koley [28]. This method aimed to improve CPU utilization, reduce makespan, and improve the makespan-deadline meeting ratio. This method was classified into 2 parts. The first part served to estimate the processing time for each task based on deadline and task dependencies. The second part was responsible for allocating VMs to reduce power consumption and increase resource utilization. The experiments showed that CDMWS outperformed the Enriched Workflow Scheduling Algorithm (EWSA) and Round Robin (RR) algorithm.

Efficient resource allocation with a focus on solving energy efficiency problems using the multi-objective optimization (MOO) method was discussed by Shrimali and Patel [29]. A VM allocation approach has also been proposed using MOO. According to the experimental results, MOO led to energy savings due to the efficient allocation of resources without affecting the performance of the data center.

Many studies have used a meta-heuristic algorithm and a hybrid meta-heuristic algorithm to solve problems in a cloud computing environment. For example, Tawfeek *et al.* [35] discussed the adoption of the Ant Colony Optimization (ACO) algorithm for reducing the execution time of tasks in a cloud computing environment. The authors demonstrated that using ACO was efficient. ACO worked better than other methods, such as the RR algorithm and FCFS algorithm. To reduce the makespan and optimize resource access in cloud computing, an improved PSO algorithm has been developed. The method used updating the weights of particles with the evolution of the number of iterations and to inject some random weights in the final stages of the PSO. The objective was to avoid local optimum solutions being generated in the final stages of PSO was proposed by Luo *et al.* [12].

Chen *et al.* [36] proposed the dynamic objective genetic algorithm (DOGA) by focusing on optimizing the execution time according to the deadline constraint to help reduce the cost of work and to work within the specified time. Amini *et al.* [37] proposed the resources allocation process for virtual machines in cloud computing using dragonfly optimization algorithm. The experimental showed that using dragonfly optimization algorithm helped improve task scheduling, load balancing, and resource allocations better than ACO algorithm and Hybrid Algorithm Based on Particle Swarm Optimization and Ant Colony Optimization Algorithm (ACO-PSO).

Li and Han [42] proposed a hybrid discrete ABC algorithm for flexible task scheduling problems in a cloud system. The objective of this approach was to minimize the maximum completion time, total workloads of all devices, and maximum device workload. In terms of reducing the completion time and balancing load in the cloud computing environment, the Heuristic Task Scheduling with Artificial Bee Colony (HABC) Algorithm was proposed to schedule and manage cloud resources by Kruekaew and Kimpan [43]. The experimental results showed that HABC with the Largest Job First heuristic algorithm (HABC_LJF) was the most efficient in scheduling and managing cloud resources.

Gan *et al.* [44] proposed job scheduling using a genetic simulated annealing algorithm. The primary purpose was to optimize the execution time of data center tasks. Basu *et al.* [14] introduced a hybrid meta-heuristic algorithm called GAACO, which was a combination of the GA algorithm and ACO algorithm to solve the task scheduling of IoT applications in a multiprocessor cloud environment. The method guaranteed appropriate convergence when tested with sizes of task graphs and number of different processors in terms of makespan and efficiency. They found that the GAACO algorithm performed better than the GA algorithm and ACO algorithm in a heterogeneous multiprocessor environment.

For a multi-objective task scheduling problem in a cloud environment that is solved using a meta-heuristic algorithm and a hybrid meta-heuristic algorithm, Alsadie [45] proposed a meta-heuristic framework for dynamic virtual machine allocation with optimized task scheduling in a cloud computing environment (MDVMA). MDVMA was a multi-objective scheduling method using a non-dominated sorting genetic algorithm to help reduce cost, makespan, and energy usage. In the experiments, theHeterogeneous Computing Scheduling Problems (HCSP) dataset was used with the CloudSim Simulator. The results showed that MDVMA was able to optimize task scheduling better than the ABC algorithm, Whale Optimization Algorithm (WOA), and PSO algorithm in terms of reducing the cost, makespan, and energy usage of the cloud data center.

Guo [34] proposed cloud computing multi-objective task scheduling optimization based on a fuzzy self-defense algorithm, which had good performance in terms of the shortest completion time, deadline violation rate, and utilization of virtual machine resources when compared with A Multi-Objective Optimization Scheduling Method Based on the Ant Colony Algorithm in Cloud Computing (PBACO) [33] and Task Scheduling Algorithm Based on RL. Zuo *et al.* [33] proposed a multi-objective optimization scheduling method in terms of efficiency and budget costs. This approach used an ant colony algorithm (PBACO) to help determine the optimal solution. The experiment was compared with the classical heuristic algorithm, Min-Min algorithm, and FCFS scheduling. PBACO was found to be superior to the other comparison methods.

He *et al.* [46] proposed Adaptive Multi-objective Task Scheduling (AMTS) to try to improve resource utilization, cost, energy consumption, and task completion time. AMTS used a PSO-based approach for multi-objective scheduling that considered process time and transmission time.

The proposed method applied the adaptive acceleration coefficient for particle diversity. After improving the PSO

TABLE 1. Simulation environment.

Type	Parameter	Value
Host	Number of Host	20
	MIPS	177,730
	Bandwidth	10 GB/s
	Storage	2 TB
	RAM	16 GB
	VM Monitor	Xen
Data Center	Number of Data Center	1
	VmScheduler	Time Shared
	Cost per Memory	0.1 - 1.0
	Cost per Storage	0.1 - 1.0
	VM Monitor	Xen
Cloudlet (Task)	Length of Tasks	1k – 900k
	Number of tasks	200-1,000
Virtual Machine (VM)	Number of VMs	5 - 100
	Processor speed	3,500 - 100,000 MIPS
	Memory	1 – 4 GB
	Bandwidth	1,000 – 10,000
	Cost per Memory	0.1 - 1.0
	Cost per Storage	0.1 - 1.0
	Cloudlet Scheduler	Time Shared
	Number of PEs	1
	VM Monitor	Xen

algorithm, AMTS was able to find the best solution for the cloud-based scheduling problem.

Jena [47] proposed an ABC-based approach for energy efficiency, processing time, cost, and computing resource utilization in the cloud computing environment. Tasks were allocated to the data center using a multi-objective ABC algorithm. An ant-based genetic algorithm was used to solve multi-objective scheduling problems to reduce latency and completion time while maximizing throughput in the cloud environment by Kumar and Venkatesan [48]. The multi-objective WOA was proposed by Reddy and Kumar [49] to develop scheduling in cloud computing. The authors tried to create a fitness-based schedule based on three conditions: quality of service, energy, and resource utilization. After considering the given parameters, the processing time and cost of the virtual machines were found to be reduced.

Madni *et al.* [50] proposed an innovative Multi-objective Cuckoo Search Optimization (MOCSO) algorithm for the resource scheduling problem in IaaS cloud computing environment. The major goal of the resource scheduling challenge was to reduce cloud user costs and enhance the performance by reducing makespan time. The simulation results showed that MOSCO algorithm outperformed Multi-objective ACO (MOACO), Multi-Objective GA (MOGA), Multi-Objective Min-Max (MOMM), and Multi-objective PSO (MOPSO) algorithm.

Pang *et al.* [51] developed an EDA-GA hybrid scheduling algorithm to solve the multi-objective task scheduling problem based on EDA (estimation of distribution algorithm) and GA (genetic algorithm). The constraints of scheduling problem in this model were scheduling performance and time. The advantages of this algorithm were the fast convergence speed and strong search ability. The algorithm was compared with EDA and GA using the CloudSim simulation experiment platform. According to the testing results, the EDA-GA hybrid algorithm effectively reduced job completion time and improved load balancing ability.

Neelima and Reddy [52] proposed a load balancing task scheduling algorithm in cloud using Adaptive Dragonfly algorithm (ADA) which was a combination of dragonfly algorithm (DA) and firefly algorithm (FA). The development of a multi-objective function was based on three parameters: completion time, processing costs, and load. The performance of this method was measured in terms of execution cost and time. When compared to DA and FA, the experimental results showed that the proposed approach achieved better load balancing results.

There is also research combining osmotic behavior with bio-inspired algorithms, for example, Gamal *et al.* [53] presented Osmotic hybrid artificial bee and ant colony (OH_BAC) algorithm which derives from the osmosis theory in the chemistry science. This algorithm was used to form osmotic computing and find load balancing for VM placement. OH_BAC used an osmosis approach to create a low-energy cloud computing environment. In this algorithm, ACO and ABC collaborated to find the optimum VM for migrating to the best physical machine (PM). To reduce power consumption, OH_BAC activated the most appropriate osmotic host among all PMs in the system. The algorithm was compared with ACO, ABC, H_BAC, and host overloading detection algorithms. The experimental results showed that when compared to other algorithms in fixed and variable loads, OH_BAC improved energy consumption, service level agreement violation (SLAV), number of virtual machine migration, and number of host shutdowns.

Machine learning algorithms have been used to solve challenging problems in cloud computing environments [54]. For example, Farahnakian *et al.* [39] proposed the Reinforcement Learning-based Dynamic Consolidation method (RL-DC) to reduce energy consumption and optimize resource usage in cloud data centers.

Caviglione *et al.* [55] introduced a deep reinforcement learning-based approach for the placement of VMs in cloud data centers. DRL was used to select the best location possible for each VM. Jena *et al.* [56] proposed a hybrid meta-heuristic algorithm called QMPSO for balancing loads between VMs in cloud computing using hybridization of modified particle swarm optimization (MPSO) and an improved Q-learning algorithm. QMPSO helped to improve the makespan, throughput, and power consumption during load balancing and effectively reduced the waiting time of tasks.

A framework for cloud resource allocation with the goal of deploying resources in green cloud was proposed by Thein *et al.* [57]. The proposed framework was based on a reinforcement learning mechanism and fuzzy logic. Its efficiency was measured by CPU utilization at the data center,

which measuring Power Usage Effectiveness (PUE) and Data Center infrastructure Efficiency (DCiE). Simulation results using CloudSim showed that this framework can achieve effective performance for high data center energy efficiency and prevent SLA violation. The goal of the task scheduling and resource allocation model by hybrid ant colony optimization and deep reinforcement learning is to reduce the completion time and improve the utilization of idle resources with the use of a binary in-order traversal tree using weighted values. For task scheduling, a DRL algorithm was used to find idle resources and ACO to find the most suitable VM for each task by Rugwiro *et al.* [41].

As mentioned previously, developing effective algorithms for task scheduling and selecting the right resources in cloud computing environments was found to be very important. Many studies put much effort into organizing tasks or allocating proper resources for each task considering single objective, bi-objective and multi-objective scheduling. Therefore, this paper proposed the multi-objective optimization of the task scheduling problem, while previous studies focused mainly on the objectives of makespan or processing time and cost because both of these objectives meet the needs of the users. However, to work in cloud computing, it is necessary to consider various conditions or objectives. In addition, load balancing in the cloud must be considered. A hybrid meta-heuristic algorithm has been introduced to help optimize performance in the cloud. Nevertheless, some algorithms are weak in local search, while some algorithms have a weakness in global search optimization. According to related studies, the ABC algorithm can be helpful in solving the multi-objective task scheduling problem because it is capable of explorative behavior. In contrast, exploitative behavior, which is a part of the onlooker bee, was weak. In this case, reinforcement learning can help solve this problem because Q-learning can improve the solution quality. Therefore, we propose a method that can solve the multi-objective task scheduling problem using the ABC algorithm and Q-learning (MOABCQ). MOABCQ helps determine the order of tasks for suitable available resources environments to find the most appropriate task scheduling solution. Moreover, MOABCQ also provides a load balancing system.

III. PROBLEM DEFINITION AND FORMULATION OF OBJECTIVE FUNCTION

Many users operate their works on the cloud environment, which makes the scheduling process play an important role in resource utilization, response time, latency, and load balancing. Task scheduling problems can be described as follows.

Input:

- Let $V = \{v_1, v_2, v_3, \ldots, v_m\}$ where V is a collection of VMs, and m is the total number of VMs in the cloud network. Each VM has its own resources (e.g., CPU, RAM, and bandwidth) and the cost of usage and the computing power are defined differently; v_i presents the i^{th} VM.
- Let $T = \{t_1, t_2, t_3, \ldots, t_n\}$ where T is the set of assigned tasks, and n is the number of independent tasks performed on the VMs (m). t_j represents the j^{th} task. Each task submission contains the number of instructions, memory required, CPU required, etc.

Output:

- Optimize scheduling, map n tasks and m VMs (resources) to minimize makespan, cost, and resource utilization and to increase load balancing in resource utilization.

In general, multi-objective optimization problem consists of several fitness functions (Objective function) as: $F(x) = [f_1(x), f_2(x), \ldots, f_{obj}(x)]$ where obj is number of objectives, and $f_i(x)$ represents i^{th} objective function. This problem does not have a single solution and a set of non-dominated solutions can be found, known as a pareto optimal solutions.

We propose the task scheduling algorithm in cloud computing using conditions of multi-objective scheduling approaches to increase the efficiency of schedule optimization and resource utilization, to maximize VM throughput, and create load balancing between VMs. The most common evaluation factors are cost, energy consumption, task completion time, task waiting time, flow time, failure rate, profit, carbon emission, makespan, and reliability [58]. The proposed method determines the suitability conditions using cost, resource utilization, and makespan as factors that help in proper scheduling. Scheduling and load balancing in a cloud computing environment requires a fitness function design to provide optimal solutions and also considers the multi-objective used to find the answer. We used weighted sum approach [59], [60] for solving the multi-objective optimization problem. This approach is used to convert a multi-objective optimization problem to a single objective with weights which represents preferences among objectives by the decision maker. We have considered three objectives in the article as follows:

1) *The first objective* is defined in the condition of makespan or task execution time, which is the time that the system completes its last task. The makespan is a useful factor for multi-objective scheduling approaches that can reduce the execution time and allow tasks to be completed prematurely. Each VM has a different execution time for completing the task determined by the makespan. If a maximum of the execution time value is high and the makespan value is also high, the system is considered poorly distributed tasks to the VMs. However, if the maximum execution time value is low, the makespan value is also low. Thus, the system can evenly distribute tasks among the resources in the system.

Considering where each task t_j $(t_j \in T)$ is assigned to the VM, v_i ($v_i \in V$) is represented by t_{ji}, so the VM task is represented by $v_i = \{t_{xi}, t_{yi}, \ldots, t_{zi}\}$.

The total execution time (ET) of task processing on v_i can be obtained by (1).

$$ET(v_i) = \sum_{t_{ji} \in v_i} ExtTime(t_{ji}) = \frac{\sum_{t_{ji} \in v_i} length(t_j)}{CPU(v_i)} \quad (1)$$

where $ExtTime\left(t_{ji}\right)$ is the execution time of t_j processing in v_i. This value can be calculated by (2).

$$ExtTime\left(t_{ji}\right) = \frac{length(t_j)}{CPU(v_i)} \tag{2}$$

where $length(t_j)$ is the length of the j^{th} task, the length of the task is defined in terms of the number of instructions (million instructions), and $CPU(v_i)$ is the CPU rate used to process the j^{th} VM in the cloud. Makespan is the maximum value of the execution time of all VMs and can be calculated by (3).

$$Makespan = Max(ExtTime(v_i)), \quad 1 \leq i \leq m \tag{3}$$

MinMakespan is a lower bound of makespan, which is the minimum time required by the system to complete all tasks. MinMakespan is calculated by (4).

$$MinMakespan = Min(ExtTime(v_i)), \quad 1 \leq i \leq m \tag{4}$$

Fitness function in terms of makespan (F_1) can be calculated by (5).

$$F_1 = \frac{MinMakespan}{Makespan} \tag{5}$$

2) *The second objective* is defined in terms of cost which is the cost of requesting to be processed by the task and can be calculated from CPU cost, memory usage cost, and bandwidth usage cost. Estimated cost when t_j is processed at v_i can be calculated by (6).

$$Cost\left(t_{ji}\right) = (c_1 * ExtTime(t_{ji})) + (c_2 * ExtTime(t_{ji})) + (c_3 * ExtTime(t_{ji})) \tag{6}$$

where c_1, c_2, c_3 are CPU usage cost per unit, memory usage cost per unit, and bandwidth usage cost per unit in v_i, respectively.

The total cost ($TCost$) is calculated as the sum of all tasks processing on all VMs, which can be calculated from (7).

$$TCost = \sum_{j=1}^{n} \sum_{i=1}^{m} Cost(t_{ji}) \tag{7}$$

MinTCost is the lowest cost when the set of assigned tasks T is processed in the VM where the VM process task t_j gives the lowest cost, called the $MinTCost(t_j)$ value, which means that MinTCost is only for task t_j and can be calculated from (8).

$$\begin{aligned} MinTCost &= \sum_{t_j \in T} MinCost\left(t_j\right) \\ &= \sum_{t_{ji} \in T} min_{1 \leq i \leq m}(Cost\left(t_{ji}\right)) \end{aligned} \tag{8}$$

The fitness function in terms of cost (F_2) can be calculated by (9)

$$F_2 = \frac{MinTCost}{TCost} \tag{9}$$

3) *The third objective* is defined in terms of the utilization of resources (CPU and memory) sent to a different number of processing units than in the cloud network. If the requested task is sent to v_i, we can calculate the memory load of v_i by (10).

$$LM_i = AM_i + \frac{RM_j}{TM_i} \tag{10}$$

where AM_i is the amount of memory usage before executing task t_j at the i^{th} VM

RM_j is the memory containing the request of the j^{th} task

TM_i is the total memory available at the i^{th} VM

The next parameter is the CPU. If the requested task is sent to v_i, we can calculate the CPU load of v_i (LC_i) using (11).

$$LC_i = AC_i + \frac{RC_j}{TC_i} \tag{11}$$

where AC_i is the amount of CPU usage before executing task t_j at the i^{th} VM

RC_j is the CPU that requests the j^{th} task

TC_i is the total CPU available at the i^{th} VM.

VM utilization evaluation (VU_i) [61] can be calculated by (12).

$$F_3 = VU_i = \frac{\omega_1}{1 - LM_i} * \frac{\omega_2}{1 - LC_i} \tag{12}$$

where ω_1 and ω_2 are the weights for the CPU and memory, respectively. $\omega_1 + \omega_2 = 1$ in this paper, given ω_1, ω_2 are both equal to 0.5 because CPU and memory are equally important.

Total load on k host (LH), where k is the total number of hosts in the system, can be calculated from (13).

$$LH_k = \sum_{i=0}^{m_k} VU_{ki} \tag{13}$$

The average load on all physical machines in cloud (AL) can be calculated by (14).

$$AL = \frac{\sum_{k=0}^{p} LH_k}{p} \tag{14}$$

where p is the number of hosts in the cloud network.

The difference in load among each host and the average load on the cloud network is calculated from $|LH_k - AL|$. The fitness function is defined in terms of the utilization of resources, which can be calculated by (15).

$$F_3 = \sum_{k=0}^{p} |LH_k - AL| \tag{15}$$

Fitness function is created by calculating the weighted average of each individual fitness function. The proposed fitness function (F) is shown in (16).

$$F = (\gamma_1 * F_1) + (\gamma_2 * F_2) + (\gamma_3 * F_3) \tag{16}$$

where γ_1, γ_2, γ_3, and $\gamma \epsilon [0, 1]$ are the balance coefficients between makespan, total cost, and resource utilization. Maximizing the utility function (F) results in a better solution.

IV. HYBRID ABC ALGORITHM FOR LOAD BALANCING

A. Q-LEARNING ALGORITHM

Q-learning [62] is one of the Reinforcement Learning (RL) algorithms. RL is one of the machine learning methods that allows agents to learn in their environment and action by changing their state to receive rewards or penalties based on the feedback obtained from the environment. The main purpose of RL is to learn the agent through trial and error between the agent and the environment. The agent is able to receive the environment situation through a state and choose an action that affects the environment to obtain the best reward and learn through past mistakes. The Markov decision process (MDP) is a framework for the decision-making of agents because of the uncertain environment, and the result is sometimes stochastic. The agent chooses to perform the same action for the same situation or state, but it may not always obtain the same result. The Q-learning algorithm process is shown in Fig. 2.

Given that the set of states $S = \{s_1, s_2, s_3, \ldots, s_n\}$ is in the environment, each state has a set of actions. The $A = \{a_1, a_2, a_3, \ldots, a_m\}$ agent selects action $a_t \epsilon A$ at time t in state $s_t \epsilon S$ to pass to the next state $s_{t+1} \epsilon S$ through the transition process and receives a reward r_{t+1} from the environment. To process the tasks, it is necessary to select the appropriate action to maximize the Q-value of each state, which is the primary objective of finding the optimal policy in cloud computing. The Q-value function depends on the selection of action in the state. Given the agent in state s_t and selecting action a_t, the Q-value function is expected to move to the best state and gain to maximize the total expected reward in the environment. The Q-value can be calculated by (17)

$$Q(s_t, a_t) = (1-\alpha)\, Q(s_t, a_t) + \alpha\, [r_t + \gamma max_{a+1} Q(s_{t+1}, a_{t+1})] \quad (17)$$

where α is the learning rate, $\gamma (0 < \gamma < 1)$ is the discount factor and effect on the successive state by the previous action, and r_t is the penalties or rewards awarded for performing actions in the state s_t. The Q-value derives from creating a Q-table that stores all possible states, Q-values, and appropriate actions. The Q-learning algorithm attempts to establish the optimal state from their experience, and the greedy algorithm is implemented in the Q-learning algorithm. Q-value can be computed using (18).

$$Q_{t+1}(s_t, a_t) = Q(s_t, a_t) + \alpha \left[r_t + \gamma max_{\acute{a_t}} [Q_t\left(\delta(s_t, a_t), \acute{a_t}\right)] - Q_t(s_t, a_t)\right] \quad (18)$$

where α is the learning rate, calculated from $\alpha = 1/(1 + \text{total number of visits to state } s_t)$, and δ is the transition function.

$max_a Q(s_{t+1}, a)$ is an estimate of optimal future value. A possible action in cloud computing involves load balancing, which allocates resources in VM and can be described in Algorithm 1.

Algorithm 1 Q-Update(S)

1. Set values for learning rate α, discount factor γ, reward r
2. Initialize Q-values
3. For $i = 1$ to n # n is the number of states
4. For $j = 1$ to p # p is the number of actions in each state
5. $Q\left(s_i, a_j\right) = 0$
6. End for
7. End for
8. Select an action a_i from $A = \{a_1, a_2, a_3, \ldots, a_m\}$ and execute it and go to next state s_{t+1}
9. Calculate the learning rate
10. Calculate the reward
11. Update Q_{t+1} by (18)
12. Repeat this step for new state until it converges

B. MODIFIED ABC ALGORITHM

The ABC algorithm [19] is a meta-heuristic optimization algorithm based on swarm intelligence. The swarm system consists of agents that communicate with other agents and their environment. In this algorithm, the goal of the agent is to find the best food source, where the food source represents a set of possible answers in the search space and each agent is represented by a bee.

In ABC, agents are categorized according to the functions of the bees and can be classified into 3 groups: employed bees, onlooker bees, and scout bees. Initially, each employed bee finds a random food source, whereas in each iteration, employed bees find a new food source near a current food source. After collecting the nectar, each employed bee assesses the best food source. Then, the bee will move to a new food source only if the bees have determined that new food sources are better than the previous one. The employed bees share information with the onlooker bees. The onlooker bees decide to choose new food sources based on information obtained from the employed bees. If any food source has much food or high quality, it will have a high chance of being chosen by onlooker bees. Then, each onlooker bee will find a new food source around it. They select the food source and move to a new food source. The number of iterations is also predetermined, meaning that if no better food source is found, the employed bees who own the selected food source become scout bees and are responsible for exploring the new food source in a new area of search space.

Each group of bees has different explorative and exploitative behaviors [63]. The explorative behavior of a search agent involves searching for a new food source in the search space to avoid a local optimum. In contrast, exploitative behavior involves searching for a better food source near the current food source. In this paper, we use Q-learning to improve our solution to provide more appropriate solutions to problems. The ABC algorithm has both strong explorative behavior and weak exploitative behavior; therefore, we aim to

optimize exploitative behavior. The process can be described as follows:

In the initialization phase, the ABC algorithm represents the location of the food source with possible solutions to the problem. Initially, the food source is randomly generated as shown in (19). Then, employed bees are associated with food sources. The initial value of the Q-table is 0.

$$x_{i,j}^{0} = x_j^{min} + rand\ (0,\ 1) * (x_j^{max} - x_j^{min}) \tag{19}$$

where x_j^{min} is the lower bound of the j^{th} optimization parameter, and x_j^{max} is the upper bound of the j^{th} optimization parameter.

In addition to the initialization phase, the ABC algorithm separates the algorithm into three sub-phases: employed bee phase, onlooker bee phase, and scout bee phase. The algorithm repeats all three phases until a certain maximum number of values is reached.

In the employed bee phase, the employed bees [64] find the location of the neighboring food source $v_{i,j}^{t}$ of the current food source ($\bar{x}_i^t$) using (20).

$$v_{i,j}^{t} = x_{i,j}^{t} + rand\ [-1,\ 1] * (x_{i,j}^{t} - x_{k,j}^{t}) \tag{20}$$

where $v_{i,j}^{t}$ is the j^{th} optimization parameter of $\bar{v}_i^t$, and k is the index of the food source.

If the new food source $\bar{v}_i^t$ returns a fitness value greater than the current food source x_i^t, $Fit\ (\bar{v}_i^t)\ >\ Fit\ (\bar{x}_i^t)$, employed bees forget the current food source and remember the new location. In this step, the Q-table (reward-penalty scheme) value is updated using (18). If the new food source provides a better fitness value, the employed bee will not only replace the current food source with the new food source but also update the Q-value by rewarding the selected new food source and penalty with the current food source.

In contrast, if the new food source does not provide a better fitness value, the new food source receives a penalty, and the current food source receives a reward. The Q-table is updated every time that an employed bee finds a suitable food source. Therefore, if the number of employed bees is Emp, Q-table will update Emp once.

In the onlooker bee phase, the onlooker bee selects the employed bee's food source from the Q-value in the Q-table. In contrast, the onlooker bee searches for the new food source using (21) and replacing the current food source with the new food source, if the new food source has a higher fitness value, then the Q-value will also be updated.

$$v_{i,j}^{t} = x_{i,k}^{t} + \emptyset_{i,j}^{t} \cdot r_j^t \cdot (x_{i,k}^{t} - x_{RFS,k}^{t}) \tag{21}$$

where $v_{i,j}^{t}$ is the optimization parameter of a neighboring food source $\bar{v}_i^t$, $\emptyset \in [-1, 1]$, and $x_{RFS,k}^{t}$ is the optimization parameter of the optimal food source caused by random selection.

Using (21) to improve exploitation instead of updating values in dimension, onlooker bees exploit current food and update all dimensions with different weight values, and in this step, Q-values (reward and penalty) are updated.

In the scout bee phase, if there are a number of unsuccessful attempts to find a better neighbor, that food source will be discarded, and the scout bee will randomly search for a new food source.

The pseudo-code of the MOABCQ method is presented in Algorithm 2.

Algorithm 2 MOABCQ Method

Initialization:
1. Initialize the population and calculate individual fitness values
2. Set up the parameter: best solution, maximum number of iterations, the population size
3. Find the best solution
4. while stopping criteria satisfied

Employed Bees Phase:
5. For each position do
6. Update position of employed bee by (20)
7. Estimate the new position
8. If fitness value of new position is better
9 Replace the current position with the new position
10 End if
11. Calculate probability and update the Q-table for select position in the onlooker bee phase
12. End for

Onlooker Bees Phase:
13. For each onlooker do
14. Select a position based on Q-value and probability
15. Update position of onlooker bee by (21)
16. Estimate the new position
17. If fitness value of new position is better
18. Replace the current position with the new position
19. End if
20. Update the Q-table using (18)
21. End for
22. Find the best solution (Q-table)

Scout Bees Phase:
23. For each position do
24. Abandon the solution that have not been updated and generate new solutions randomly
25. update the Q-table using (18)
26. End for
27. End while

V. EXPERIMENTAL EVALUATION AND DISCUSSIONS

In this section, the parameter settings and experimental results are described. To evaluate the effectiveness of the proposed methodology (MOABCQ), we compared it with well-known heuristic task scheduling algorithms, such as Max–Min task scheduling algorithm [22], First Come First Serve (FCFS) algorithm, and Largest Job First (LJF) algorithm. Moreover, we compared MOABCQ method with the

TABLE 2. Parameter settings of meta-heuristic algorithms.

Algorithms	Parameter	Values
HABC [43]	Number of population (n)	960
	Number of sites selected out of n visited sites (m)	96
	Number of best sites (e)	1
	Number of Employed bees	192
	Number of Onlooker bees	768
	Maximum iterations	1000
MOABCQ	Number of population (n)	960
	Number of sites selected out of n visited sites (m)	96
	Number of best sites (e)	1
	Number of Employed bees	192
	Number of Onlooker bees	768
	Maximum iterations	1000
MOPSO [67]	Number of particle size	100
	Weight (w_{min}, w_{max})	0.1, 0.9
	Self-recognition coefficients (c_1, c_2)	1.49445
	Maximum iterations	1000
MOCS [50]	Number of population size	20
	Abandon probability (P_a)	0.25
	Step size (λ)	0.01, 1
	Maximum iterations	1000

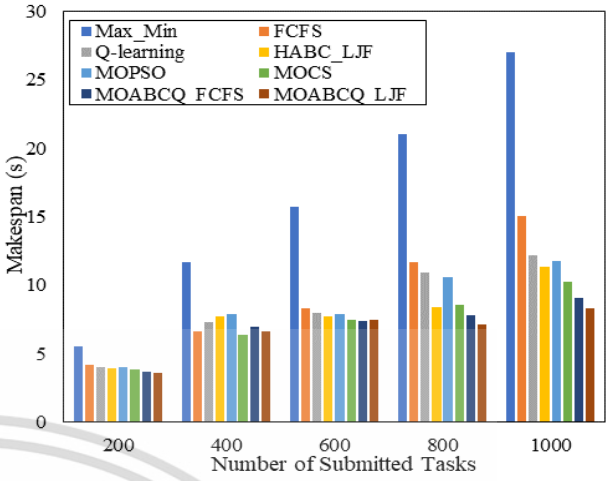
popular meta-heuristic task scheduling algorithms such as the PSO algorithm and CS algorithm, and with our previous method called Heuristic Task Scheduling with Artificial Bee Colony algorithm and largest job first (HABC_LJF) [43].

We considered evaluating the performance of the proposed method, which consists of a simulation environment, a benchmark datasets, parameter settings for the proposed method and the comparison algorithms, experimental results, and the time complexity of MOABCQ method.

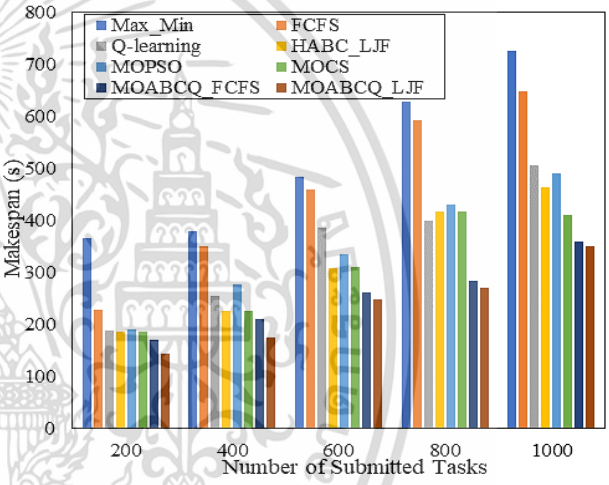
A. SIMULATION ENVIRONMENT

This section presents an experiment conducted to evaluate the performance of the proposed method (MOABCQ) when comparing task scheduling with other methods in a heterogeneous cloud computing environment. In this paper, a simulation was designed and developed using the CloudSim 3.0.3 simulator [65]. CloudSim is the most widely used simulator to implement clouds. CloudSim is a tool that can simulate virtual resources, and CloudSim can also support modeling, simulation, and experimentation of virtualized cloud-based data. This experiment was simulated on a computer with an Intel Core i7-8750H CPU (clock speed of 2.20 GHz) and 16 GBs of RAM.

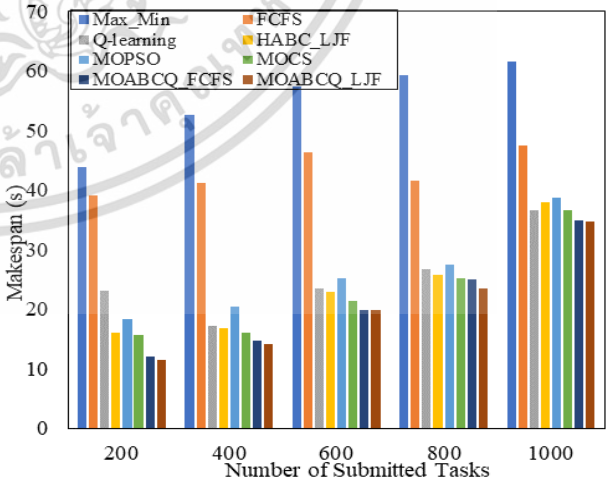
In this experiment, a virtual environment was simulated to demonstrate the efficiency of the proposed method in terms of scheduling and load balancing in a cloud computing environment. The simulation environment of this experiment was defined as shown in Table 1.



(a) Makespan comparison using the random dataset



(b) Makespan comparison using the GoCJ dataset



(c) Makespan comparison using the Synthetic workload dataset

FIGURE 3. Comparison of the performance in terms of makespan on various datasets.

B. BENCHMARK DATASETS

To evaluate the scheduling efficiency of the proposed method, three different datasets were used: 1) Random dataset, 2) Google Cloud Jobs (GoCJ) dataset [66], and 3) Synthetic workload dataset [26], which are described as:

TABLE 3. Comparison of the performance in terms of ARUR on various datasets.

Dataset	Task Scheduling Approach							
	Max-Min	FCFS	Q-learning	HABC_LJF	MOPSO	MOCS	MOABCQ_FCFS	MOABCQ_LJF
Random	0.200	0.332	0.660	0.642	0.637	0.698	0.762	**0.812**
GoCJ	0.190	0.252	0.610	0.692	0.684	0.702	0.742	**0.792**
Synthetic workload	0.178	0.505	0.755	0.723	0.722	0.761	0.796	**0.801**

1) RANDOM DATASET

We generated task sizes ranging from 1k – 70k Million Instructions (MIs). The randomly generated dataset contains a total of 1000 tasks. The dataset contains task size, number of requested CPUs, and amount of RAM being requested.

2) GOCJ DATASET

GoCJ dataset is considered a Google-like realistic dataset generated from the workload behaviors witnessed in Google cluster traces using bootstrapped Monte Carlo, a well-known simulation method. The task sizes in the GoCJ dataset range from 15k – 900k MIs, and the datasets are classified as: small size jobs (15k – 55k MIs), medium size jobs (59k – 99k MIs), large size jobs (101k – 135k MIs), extra-large size jobs (150k – 337.5k MIs), and huge size jobs (525k – 900k MIs).

3) SYNTHETIC WORKLOAD DATASET

Synthetic workload dataset iscreated by random-number generator mechanism using Monte Carlo simulation method. It consists of different tasks sizes from 1 – 45K MIs which are tiny size jobs (1-250 MIs), small size jobs (800–1200 MIs), medium size jobs (1800–2500 MIs), large size jobs (7k–10k MIs), and extra-large size jobs (30k–45k MIs).

C. PARAMETER SETTINGS FOR THE PROPOSED METHOD AND THE COMPARISON ALGORITHMS

In this experiment, we defined population parameter and other conditions from related articles which are: HABC algorithm [43], PSO algorithm [67], and CS algorithm [56]. As it is already recognized that parameter setting has effects on the efficiency of algorithms and it depends on size or nature of the problem. For this reason, tuning the parameters must be done to ensure that we use the appropriate parameters for the problem type and dataset. We did not claim, however, that the proposed method or its parameters outperform alternative algorithms for all type of problems and datasets. In the experiment, the parameters of the ABC algorithm proposed by Kruekaew and Kimpan [43] are defined in Table 2. In addition, to compare the scheduling performance, the proposed method was compared with the HABC algorithm, MOPSO algorithm, and MOCS algorithm. The parameters are defined as shown in Table 2 which based on the original papers.

D. EXPERIMENTAL RESULTS

This section describes an experimental evaluation of a proposed scheduling approach using the benchmark dataset to compare experimental results in terms of makespan, throughput, ARUR [26], cost, and DI. In this experiment, we assessed the effectiveness of the proposed method (MOABCQ). We combined the MOABCQ method with the First Come First Serve (FCFS) heuristic task scheduling called "MOABCQ_FCFS" and Largest Job First (LJF) heuristic task scheduling called "MOABCQ_LJF" and compared them with other well-known algorithms: FCFS scheduling algorithm, Max–Min task scheduling algorithm [22], Heuristic Task Scheduling with Artificial Bee Colony algorithm and largest job first (HABC_LJF) [43], Q-learning algorithm, multi-objective particle swarm optimization (MOPSO) algorithm, and multi-objective cuckoo search (MOCS) algorithm. Each dataset was run for 20 rounds, and the average of the results is proposed in the following section.

The first section presents a comparison of the performance of the proposed method in terms of makespan. This experiment was assigned to 100 VMs, and 200, 400, 600, 800, and 1000 tasks were assigned to the system. In the experiment, we used the 3 datasets mentioned earlier. The experimental results are shown in Fig. 3. According to the experimental results in Fig. 3(a), when the random dataset was tested, the MOABCQ method was found to reduce the average makespan better than the Max-Min method, FCFS, HABC_LJF, Q-learning, MOPSO, and MOCS. However, when 400 datasets were tested, MOCS gave the lowest average makespan compared to the other methods. MOCS took 3.69%, 3.77%, 8.82%, 13.85%, 20.77% 23.41%, and 82.39% less time to complete than FCFS, MOABCQ_LJF, MOABCQ_FCFS, Q-learning, HABC_LJF, MOPSO, and Max-Min, respectively.

If we consider the insight of MOABCQ, after comparing MOABCQ_FCFS and MOABCQ_LJF, MOABCQ_LJF can be found to give a lower average makespan than MOABCQ_FCFS in the case of 200, 800, and 1000 tasks. However, in the case of 600 tasks, MOABCQ_FCFS had an average makespan 0.51% less than MOABCQ_LJF.

Considering the experimental results in Fig. 3(b), in which the GoCJ dataset was used, and Fig. 3(c), in which the Synthetic workload dataset was used, MOABCQ_LJF gave the lowest average makespan. When using the GoCJ dataset, MOABCQ_LJF gave average makespan approximately 117.80%, 92.15%, 46.17%, 42.25%, 34.94%, 30.62%, and 8.21% less than the makespan of Max-Min, FCFS, Q-learning, MOPSO, HABC_LJF, MOCS, and

MOABCQ_FCFS, respectively. When using the Synthetic workload dataset, MOABCQ_LJF yielded average makespan of approximately 150.75%, 98.61%, 25.53%, 22.87%, 15.35%, 10.76%, and 2.97% less than the makespan of Max-Min, FCFS, MOPSO, Q-learning, HABC_LJF, MOCS, and MOABCQ_FCFS, respectively. It can be indicated that MOABCQ_LJF outperformed the other methods in running both datasets with all test conditions except the percentage of the average makespan reduction.

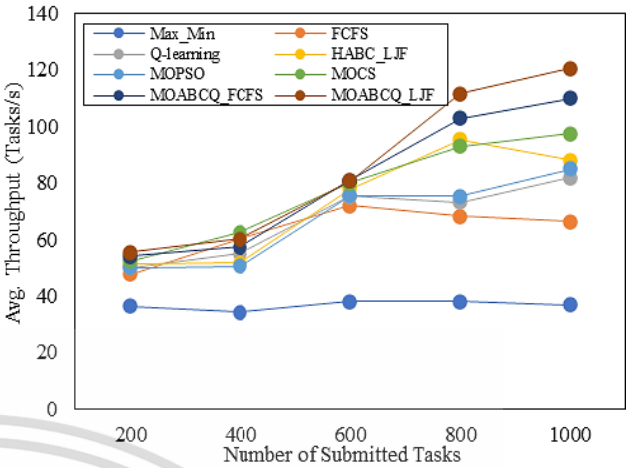
However, overall examinations showed that the proposed method can provide the lowest makespan value because MOABCQ_LJF can allocate the task to the appropriate resource. According to all these results, we can conclude that the proposed method has the potential to efficiently allocate resources in the system.

The second section presents a comparison of the efficiency of the method presented in terms of throughput, which is the number of tasks executed per unit of time. The 3 previously used datasets were used in this experiment. The experimental results of testing the efficiency with a random dataset are shown in Fig. 4(a), which indicates that MOABCQ had better throughput than other algorithms. However, when testing in 400 tasks, MOCS gave greater values than the others at 3.56%, 3.63%, 8.10%, 12.16%, 17.20%, 18.97%, and 45.17% when compared to FCFS, MOABCQ_LJF, MOABCQ_FCFS, Q-learning, HABC_LJF, MOPSO, and Max-Min, respectively. Considering the MOABCQ itself in depth, MOABCQ_LJF has a higher throughput than MOABCQ_FCFS in 200, 800, and 1000 tasks. In the case of 600 tasks, MOABCQ_FCFS provided 0.50% higher throughput than MOABCQ_LJF.

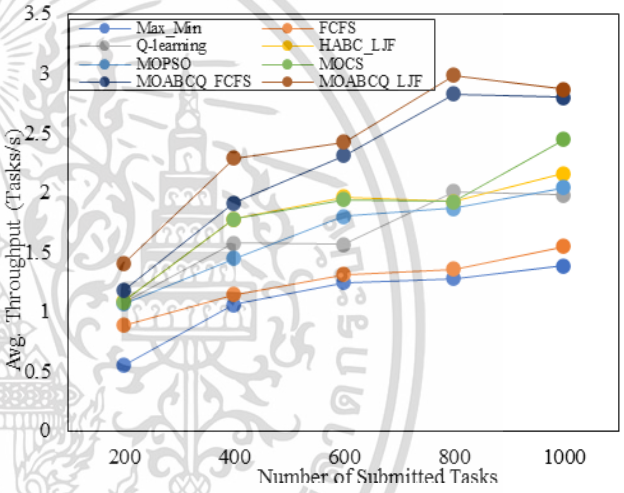
After using the GoCJ and Synthetic workload datasets to test throughput, we found that MOABCQ had better performance than the other algorithms. The results are shown in Fig. 4(b) and 4(c). When MOABCQ_LJF and MOABCQ_FCFS were considered, MOABCQ_LJF provides throughput approximately 6.14% more than MOABCQ_FCFS on average. We can conclude from the overall throughput test experiments that MOABCQ_LJF has the potential to allocate the tasks to appropriate resources in the same direction as the testing result in the first section.

The third section presents a comparison of the efficiency of the proposed method in terms of Average Resource Utilization Ratio (ARUR), which is another important condition for task scheduling in the system. In this experiment, we used 1000 tasks, 100 machines of VMs, and 3 dataset tests. The experimental results are shown in Table 3.

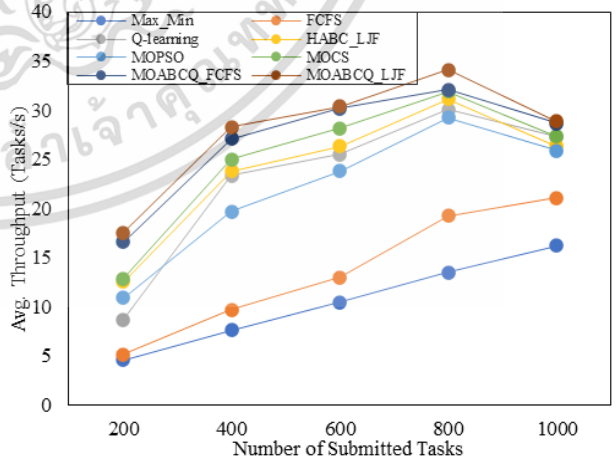
We found that MOABCQ gave higher ARUR values than the other methods. To consider in depth all 3 datasets, the algorithms used to perform in this experiment that gave the similarity of ARUR results were found to be Q-learning, HABC_LJF, MOPSO, MOCS, MOABCQ_FCFS, and MOABCQ_LJF, unlike Max-Min and FCFS. When testing with the random dataset, MOABCQ_LJF provided larger ARUR values of 6.15%, 14.01%, 18.72%, 20.94%, and 21.53% compared to MOABCQ_FCFS, MOCS, Q-learning, HABC_LJF, and MOPSO, respectively.



(a) Throughput comparison using the random dataset



(b) Throughput comparison using the GoCJ dataset



(c) Throughput comparison using the Synthetic workload dataset

FIGURE 4. Comparison of the performance in terms of throughput on various datasets.

When using the GoCJ dataset, MOABCQ_LJF gave greater ARUR values then the others at 6.31%, 11.34%, 12.63%, 13.61%, and 22.99% when compared to MOABCQ_FCFS, MOCS, HABC_LJF, MOPSO, and Q-learning. After experimenting with the Synthetic workload dataset, MOABCQ_LJF gave higher ARUR values than the MOABCQ_FCFS, MOCS, Q-learning, HABC_LJF, and MOPSO methods at 0.62%, 4.99%, 5.74%, 9.74%, and 9.86%, respectively.

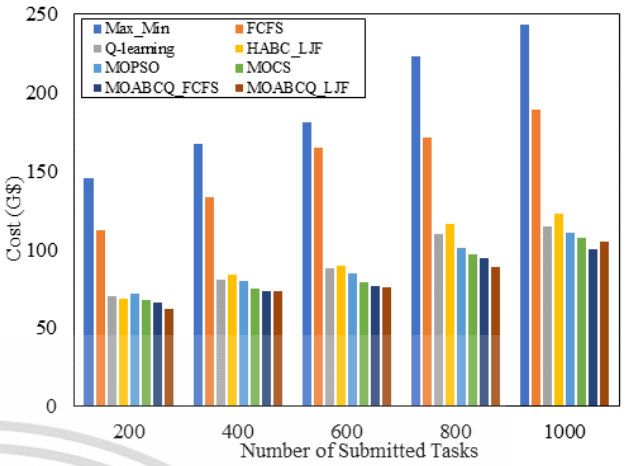
The testing with all three datasets revealed that MOABCQ_LJF provides the highest ARUR value compared to the other methods. Thus, we can conclude that the MOABCQ_LJF method can efficiently schedule tasks in the system and can help equally distribute tasks across available resources, which can help the system stay in balance mode.

The fourth section presents a comparison of the performance of the proposed method in terms of the degree of imbalance (DI) to assess the load balancing of the system. The experiments were conducted with the same datasets as in the previous experiment sections. These experiments were tested on 100 VMs and 200, 400, 600, 800, and 1000 tasks. The proposed method (MOABCQ) was compared with Max-Min, FCFS, Q-learning, HABC_LJF, MOPSO, and MOCS. The results are shown in Table 4. When testing on a random dataset and GoCJ dataset, the DI values of MOABCQ_LJF were the lowest, indicating that MOABCQ_LJF can distribute tasks more equally to existing resources in the system than the other methods. However, when using the Synthetic workload dataset, we found that in the case of setting tasks in the system equal to 200, 600, 800 and 1000 tasks, MOABCQ_LJF gave a lower DI value than the other methods. Unless testing with 400 tasks, MOABCQ_FCFS had the lowest DI value. When compared to MOABCQ_LJF, MOABCQ_FCFS can distribute tasks better than MOABCQ_LJF at 4.37%.

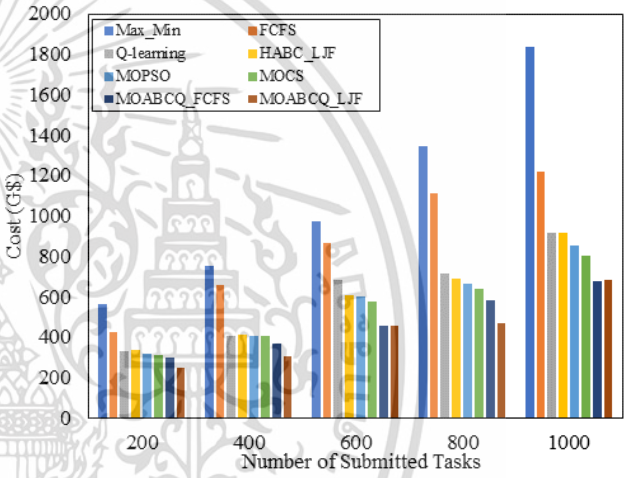
According to the testing with all three datasets, the MOABCQ method was found to be able to equally distribute the task to the available resources in the system, which resulted in a low DI value. If we consider the proposed method in depth, it reveals that MOABCQ_LJF performed more efficiently than the other compared methods. However, it depends on the dataset being tested.

The final section presents a comparison of the performance of the proposed method from a cost perspective to assess the costs or overheads when accessing cloud computing by executing 3 datasets. The experimental results are shown in Fig. 5. Considering Fig. 5(a), using the random dataset, MOABCQ was found to be able to reduce cost more than the MOCS, MOPSO, Q-learning, HABC_LJF, FCFS, and Max-Min. When MOABCQ_FCFS was compared with MOABCQ_LJF, it indicated that MOABCQ_LJF has lower cost than MOABCQ_FCFS by approximately 3.38%. However, with 1000 tasks, the MOABCQ_LJF algorithm costs 4.88% more than the MOABCQ_FCFS algorithm.

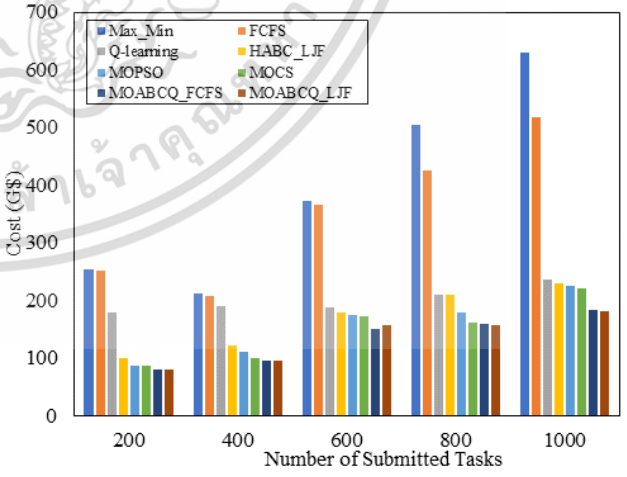
When testing with the GoCJ dataset, the results in Fig. 5(b) show that MOABCQ was able to reduce costs more than the other methods in the same way as when using the random dataset. When comparing MOABCQ_FCFS with MOABCQ_LJF and testing on 200, 400, and 800 tasks,



(a) Cost comparison using the random dataset



(b) Cost comparison using the GoCJ dataset



(c) Cost comparison using the Synthetic workload dataset

FIGURE 5. Comparison of the performance in terms of cost on various dataset.

TABLE 4. Comparison of the performance in terms of DI on various datasets.

Dataset	Task	Task Scheduling Approach							
		Max-Min	FCFS	Q-learning	HABC_LJF	MOPSO	MOCS	MOABCQ_FCFS	MOABCQ_LJF
Random	200	1.285	0.577	0.442	0.462	0.334	0.280	0.264	**0.200**
	400	0.902	0.425	0.285	0.241	0.309	0.177	0.177	**0.166**
	600	0.916	0.685	0.585	0.378	0.555	0.203	0.134	**0.116**
	800	1.138	0.660	0.237	0.157	0.573	0.177	0.137	**0.115**
	1000	1.330	0.526	0.297	0.175	0.221	0.159	0.097	**0.093**
GoCJ	200	1.023	0.829	0.311	0.430	0.450	0.250	0.270	**0.132**
	400	0.997	0.944	0.272	0.283	0.278	0.204	0.146	**0.123**
	600	1.114	0.652	0.299	0.291	0.299	0.281	0.276	**0.176**
	800	1.234	0.544	0.478	0.294	0.499	0.287	0.387	**0.275**
	1000	1.112	0.912	0.355	0.373	0.387	0.363	0.314	**0.251**
Synthetic workload	200	1.529	0.964	0.225	0.410	0.412	0.378	0.268	**0.187**
	400	1.314	0.905	0.342	0.463	0.472	0.315	**0.293**	0.305
	600	1.130	0.766	0.302	0.734	0.742	0.301	0.418	**0.214**
	800	1.147	0.670	0.295	0.368	0.413	0.274	0.141	**0.117**
	1000	0.834	0.679	0.211	0.235	0.337	0.210	0.209	**0.122**

we found that the MOABCQ_LJF algorithm has a lower cost than the MOABCQ_FCFS algorithm at approximately 20.79%. In addition, when testing with 600 and 1000 tasks, MOABCQ_FCFS was found to have a higher cost than the MOABCQ_LJF method by 0.48% on average.

When testing with the Synthetic workload dataset, the results in Fig. 5(c) show that the MOABCQ method has a lower cost than the other comparison methods in the same way as when testing with the previous two datasets. After comparing MOABCQ_FCFS and MOABCQ_LJF, in the case of setting tasks in the system equal to 200, 600, 800, and 1000 tasks, the MOABCQ_LJF algorithm was found to have lower cost than MOABCQ_FCFS, except for 400 tasks, MOABCQ_FCFS has lower cost than MOABCQ_LJF method at 1.90%.

According to the testing with all three datasets, the proposed method of MOABCQ was able to reduce costs more than the other comparison methods. However, when comparing MOABCQ_LJF and MOABCQ_FCFS, we found that MOABCQ_LJF can be more appropriately used for task scheduling with existing resources in the system than MOABCQ_FCFS. Considering the difference in the percentage between the two methods, MOABCQ_LJF has a smaller percentage. However, it depends on the dataset to be tested.

E. THE TIME COMPLEXITY OF MOABCQ METHOD

The time complexity of MOABCQ method can be calculated as: in ABC, an initial population of n is given and the bee is classified into Employed bees and Onlooker bee. Therefore, the number of iterations to find suitable VMs in the cloud is n and the number of updated data in the Q-table is n as well. As a result, the time complexity of MOABCQ is $O(n)$. If ABC repeats this step k times, the time complexity is equal to $k \times O(n)$. Since k is a constant, the total time complexity of MOABCQ is equal to $O(n)$.

VI. CONCLUSION

In this article, we propose the multi-objective optimization scheduling method in heterogeneous cloud computing using the MOABCQ method. This method considered the selection of suitable VMs based on calculating the fitness of each VM. Heuristic approaches which are FCFS and LJF were also included. The experiments were conducted with various datasets to observe the performance of the proposed algorithms. The proposed method helps load balancing tasks with existing resources in the system and also improves makespan reduction, DI reduction, cost reduction, and throughput and ARUR increases when compared to the Max-Min, FCFS, Q-learning, HABC_LJF, MOPSO, and MOCS algorithms. The experimental results indicated that the proposed method outperformed the others. However, we cannot guarantee that the MOABCQ_LJF algorithm is optimal. Nevertheless, the performance of the system cannot be optimized in every test dataset.

Task scheduling in a multi-cloud, fog cloud, or edge cloud environment can be challenging and interesting work in the future. We propose a scheduling arrangement method in different environments. Other machine learning algorithms may also be applied further. In addition, the proposed method can also be tested in a real-world environment to observe the performance of the MOABCQ method.

REFERENCES

[1] T. S. George and V. P. S. Kumar, "Multicloud computing for on-demand resource provisioning using clustering," in *Proc. 3rd Int. Conf. Sustain. Energy Intell. Syst. (SEISCON)*, 2012, pp. 435–440.

[2] S. Yang, L. Pan, Q. Wang, S. Liu, and S. Zhang, "Subscription or pay-as-you-go: Optimally purchasing IaaS instances in public clouds," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2018, pp. 219–226.

[3] D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, and W. Wang, "Quality-of-service in cloud computing: Modeling techniques and their applications," *J. Internet Services Appl.*, vol. 5, pp. 1–17, Dec. 2014.

[4] K. Psychas and J. Ghaderi, "On non-preemptive VM scheduling in the cloud," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 1, no. 2, pp. 1–29, Dec. 2017, doi: 10.1145/3154493.

[5] S. Crago, K. Dunn, P. Eads, L. Hochstein, D. Kang, M. Kang, D. Modium, K. Singh, J. Suh, and J. Walters, "Heterogeneous cloud computing," in *Proc. IEEE Int. Conf. Clust. Comput.*, Austin, TX, USA, Feb. 2011, pp. 378–385, doi: 10.1109/CLUSTER.2011.49.

[6] J. W. M. Bush, B. A. Thurber, and F. Blanchette, "Particle clouds in homogeneous and stratified environments," *J. Fluid Mech.*, vol. 489, pp. 29–54, Jul. 2003, doi: 10.1017/S0022112003005160.

[7] R. Messier, "Virtual servers and platform as a service," in *Proc. Collaboration Cloud Comput. Secur., Social Media, Unified Commun.*, 2014, pp. 77–91, doi: 10.1016/B978-0-12-417040-7.00005-8.

[8] O. Alsaryrah, I. Mashal, and T.-Y. Chung, "Bi-objective optimization for energy aware Internet of Things service composition," *IEEE Access*, vol. 6, pp. 26809–26819, 2018, doi: 10.1109/ACCESS.2018.2836334.

[9] L. Liu, M. Zhang, R. Buyya, and Q. Fan, "Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing," *Concurr. Comput. Pract. Exp.*, vol. 29, no. 5, p. e3942, 2017, doi: 10.1002/cpe.3942.

[10] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang, "A market-oriented hierarchical scheduling strategy in cloud workflow systems," *J. Supercomput.*, vol. 63, no. 1, pp. 256–293, 2013, doi: 10.1007/s11227-011-0578-4.

[11] D. Yagyasen, M. Darbari, P. K. Shukla, and V. K. Singh, "Diversity and convergence issues in evolutionary multiobjective optimization: Application to agriculture science," *IERI Proc.*, vol. 5, pp. 81–86, Oct. 2013, doi: 10.1016/j.ieri.2013.11.074.

[12] F. Luo, Y. Yuan, W. Ding, and H. Lu, "An improved particle swarm optimization algorithm based on adaptive weight for task scheduling in cloud computing," in *Proc. 2nd Int. Conf. Comput. S. App. Eng.*, 2018, pp. 1–5, doi: 10.1145/3207677.3278089.

[13] I. Alharkan, M. Saleh, M. A. Ghaleb, H. Kaid, A. Farhan, and A. Almarfadi, "Tabu search and particle swarm optimization algorithms for two identical parallel machines scheduling problem with a single server," *J. King Saud Univ.-Eng. Sci.*, vol. 32, no. 5, pp. 330–338, Jul. 2020, doi: 10.1016/j.jksues.2019.03.006.

[14] S. Basu, M. Karuppiah, K. Selvakumar, and K. Li, "An intelligent/cognitive model of task scheduling for IoT applications in cloud computing environment," *Future Gener. Comput. Syst.*, vol. 88, pp. 254–261, Nov. 2018, doi: 10.1016/j.future.2018.05.056.

[15] J. Horn, N. Nafpliotis, and D. E. Goldberg, "A niched Pareto genetic algorithm for multiobjective optimization," in *Proc. 1st IEEE Conf. Evol. Comput. World Congr. Comput. Intell.*, vol. 1, Jun. 1994, pp. 82–87, doi: 10.1109/ICEC.1994.350037.

[16] J. Knowles and D. Corne, "Approximating the nondominated front using the Pareto archived evolution strategy," *Evol. Comput.*, vol. 8, no. 2, pp. 149–172, Jan. 2000, doi: 10.1162/106365600568167.

[17] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," ERU, Kayseri, Turkey, Tech. Rep.-tr06, Oct. 2005.

[18] B. Akay and D. Karaboga, "A modified artificial bee colony algorithm for real-parameter optimization," *Inf. Sci.*, vol. 192, no. 1, pp. 120–142, Apr. 2012, doi: 10.1016/j.ins.2010.07.015.

[19] D. Karaboga and B. Gorkemli, "A combinatorial artificial bee colony algorithm for traveling salesman problem," in *Proc. Int. Symp. Innov. Intell. Syst. Appl.*, Jun. 2011, pp. 50–53.

[20] X. Li, D. Peng, B. Du, J. Guo, W. Xu, and K. Zhuang, "Hybrid artificial bee colony algorithm with a rescheduling strategy for solving flexible job shop scheduling problems," *Comput. Ind. Eng.*, vol. 113, pp. 10–26, Nov. 2017, doi: 10.1016/j.cie.2017.09.005.

[21] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, 1975.

[22] Y. Mao, X. Chen, and X. Li, "Max-min task scheduling algorithm for load balance in cloud computing," in *Proc. Int. Conf. Comput. Sci. Inf. Technol.*, 2014, pp. 457–465.

[23] T. Islam and M. S. Hasan, "A performance comparison of load balancing algorithms for cloud computing," in *Proc. Int. Conf. Frontiers Adv. Data Sci. (FADS)*, Oct. 2017, pp. 130–135.

[24] G. Patel, R. Mehta, and U. Bhoi, "Enhanced load balanced min-min algorithm for static meta task scheduling in cloud computing," *Proc. Comput. Sci.*, vol. 57, pp. 545–553, Jan. 2015, doi: 10.1016/j.procs.2015.07.385.

[25] H. Zhang, J. Shi, B. Deng, G. Jia, G. Han, and L. Shu, "MCTE: Minimizes task completion time and execution cost to optimize scheduling performance for smart grid cloud," *IEEE Access*, vol. 7, pp. 134793–134803, 2019, doi: 10.1109/ACCESS.2019.2942067.

[26] A. Hussain, M. Aleem, A. Khan, M. A. Iqbal, and M. A. Islam, "RALBA: A computation-aware load balancing scheduler for cloud computing," *Cluster Comput.*, vol. 21, no. 3, pp. 1667–1680, 2018, doi: 10.1007/s10586-018-2414-6.

[27] B. Mondal, K. Dasgupta, and P. Dutta, "Load balancing in cloud computing using stochastic hill climbing—A soft computing approach," *Proc. Technol.*, vol. 4, pp. 783–789, Jun. 2012, doi: 10.1016/j.protcy.2012.05.128.

[28] M. Adhikari and S. Koley, "Cloud computing: A multi-workflow scheduling algorithm with dynamic reusability," *Arabian J. Sci. Eng.*, vol. 43, no. 2, pp. 645–660, Feb. 2018, doi: 10.1007/s13369-017-2739-0.

[29] B. Shrimali and H. Patel, "Multi-objective optimization oriented policy for performance and energy efficient resource allocation in cloud environment," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 32, no. 7, pp. 860–869, Sep. 2020, doi: 10.1016/j.jksuci.2017.12.001.

[30] C.-W. Tsai and J. J. P. C. Rodrigues, "Metaheuristic scheduling for cloud: A survey," *IEEE Syst. J.*, vol. 8, no. 1, pp. 279–291, Mar. 2014, doi: 10.1109/JSYST.2013.2256731.

[31] M. Kalra and S. Singh, "A review of metaheuristic scheduling techniques in cloud computing," *Egyptian Informat. J.*, vol. 16, no. 3, pp. 275–295, 2015, doi: 10.1016/j.eij.2015.07.001.

[32] F. Ramezani, J. Lu, J. Taheri, and F. K. Hussain, "Evolutionary algorithm-based multi-objective task scheduling optimization model in cloud environments," *World Wide Web*, vol. 18, no. 6, pp. 1737–1757, 2015, doi: 10.1007/s11280-015-0335-3.

[33] L. Zuo, L. Shu, S. Dong, C. Zhu, and T. Hara, "A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing," *IEEE Access*, vol. 3, pp. 2687–2699, 2015, doi: 10.1109/ACCESS.2015.2508940.

[34] X. Guo, "Multi-objective task scheduling optimization in cloud computing based on fuzzy self-defense algorithm," *Alexandria Eng. J.*, vol. 60, no. 6, pp. 5603–5609, Dec. 2021.

[35] M. A. Tawfeek, A. El-Sisi, A. E. Keshk, and F. A. Torkey, "Cloud task scheduling based on ant colony optimization," in *Proc. 8th Int. Conf. Comput. Eng. Syst. (ICCES)*, Nov. 2013, pp. 64–69.

[36] Z. Chen, K. Du, Z. Zhan, and J. Zhang, "Deadline constrained cloud computing resources scheduling for cost optimization based on dynamic objective genetic algorithm," in *Proc. CEC*, 2015, pp. 708–714, doi: 10.1109/CEC.2015.7256960.

[37] Z. Amini, M. Maeen, and M. R. Jahangir, "Providing a load balancing method based on dragonfly optimization algorithm for resource allocation in cloud computing," *Int. J. Netw. Distrib. Comput.*, vol. 6, no. 1, pp. 35–42, 2018, doi: 10.2991/ijndc.2018.6.1.4.

[38] M. S. Sanaj and P. M. Joe Prathap, "An efficient approach to the map-reduce framework and genetic algorithm based whale optimization algorithm for task scheduling in cloud computing environment," *Mater. Today, Process.*, vol. 37, pp. 3199–3208, Oct. 2021.

[39] F. Farahnakian, P. Liljeberg, and J. Plosila, "Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning," in *Proc. 22nd Euromicro Int. Conf. Parallel Distrib. Netw.-Based Process.*, Turin, Italy, 2014, pp. 500–507.

[40] S. Ismaeel, R. Karim, and A. Miri, "Proactive dynamic virtual-machine consolidation for energy conservation in cloud data centres," *J. Cloud Comput.*, vol. 7, no. 1, pp. 1–28, Dec. 2018, doi: 10.1186/s13677-018-0111-x.

[41] U. Rugwiro, C. Gu, and W. Ding, "Task scheduling and resource allocation based on ant-colony optimization and deep reinforcement learning," *J. Internet Technol.*, vol. 20, no. 5, pp. 1463–1475, 2019, doi: 10.3966/160792642019092005013.

[42] J.-Q. Li and Y.-Q. Han, "A hybrid multi-objective artificial bee colony algorithm for flexible task scheduling problems in cloud computing system," *Cluster Comput.*, vol. 23, no. 4, pp. 2483–2499, Dec. 2020, doi: 10.1007/s10586-019-03022-z.

[43] B. Kruekaew and W. Kimpan, "Enhancing of artificial bee colony algorithm for virtual machine scheduling and load balancing problem in cloud computing," *Int. J. Comput. Intell. Syst.*, vol. 13, no. 1, pp. 496–510, 2020, doi: 10.2991/ijcis.d.200410.002.

[44] G.-N. Gan, T.-L. Huang, and S. Gao, "Genetic simulated annealing algorithm for task scheduling based on cloud computing environment," in *Proc. Int. Conf. Intell. Comput. Integr. Syst.*, Oct. 2010, pp. 60–63.

[45] D. Alsadie, "A metaheuristic framework for dynamic virtual machine allocation with optimized task scheduling in cloud data centers," *IEEE Access*, vol. 9, pp. 74218–74233, 2021, doi: 10.1109/ACCESS.2021.3077901.

[46] H. He, G. Xu, S. Pang, and Z. Zhao, "AMTS: Adaptive multi-objective task scheduling strategy in cloud computing," *China Commun.*, vol. 13, no. 4, pp. 162–171, Apr. 2016, doi: 10.1109/CC.2016.7464133.
[47] R. Jena, "Task scheduling in cloud environment: A multi-objective ABC framework," *J. Inf. Optim. Sci.*, vol. 38, pp. 1–19, Jan. 2017, doi: 02522667.2016.1250460.
[48] A. Kumar and M. Venkatesan, "Multi-objective task scheduling using hybrid genetic-ant colony optimization algorithm in cloud environment," *Wireless Pers. Commun.*, vol. 107, pp. 1835–1848, 2019, doi: 10.1007/s11277-019-06360-8.
[49] G. N. Reddy and S. P. Kumar, "Multi objective task scheduling algorithm for cloud computing using whale optimization technique," in *Proc. Int. Conf. Next Gener. Comput., Technol.* Singapore: Springer, 2017, pp. 286–297.
[50] S. H. H. Madni, M. S. A. Latiff, J. Ali, and S. M. Abdulhamid, "Multi-objective-oriented cuckoo search optimization-based resource scheduling algorithm for clouds," *Arabian J. Sci. Eng.*, vol. 44, no. 4, pp. 3585–3602, 2019, doi: 10.1007/s13369-018-3602-7.
[51] S. Pang, W. Li, H. He, Z. Shan, and X. Wang, "An EDA-GA hybrid algorithm for multi-objective task scheduling in cloud computing," in *IEEE Access*, vol. 7, pp. 146379–146389, 2019, doi: 10.1109/ACCESS.2019.2946216.
[52] P. Neelima and A. R. M. Reddy, "An efficient load balancing system using adaptive dragonfly algorithm in cloud computing," *Cluster Comput.*, vol. 23, pp. 2891–2899, 2020, doi: 10.1007/s10586-020-03054-w.
[53] M. Gamal, R. Rizk, H. Mahdi, and B. E. Elnaghi, "Osmotic bio-inspired load balancing algorithm in cloud computing," *IEEE Access*, vol. 7, pp. 42735–42744, 2019, doi: 10.1109/ACCESS.2019.2907615.
[54] U. A. Butt, M. Mehmood, S. B. H. Shah, R. Amin, M. W. Shaukat, S. M. Raza, D. Y. Suh, and M. J. Piran, "A review of machine learning algorithms for cloud computing security," *Electronics*, vol. 9, no. 9, p. 1379, Aug. 2020, doi: 10.3390/electronics9091379.
[55] L. Caviglione, M. Gaggero, M. Paolucci, and R. Ronco, "Deep reinforcement learning for multi-objective placement of virtual machines in cloud datacenters," *Soft. Comput.*, vol. 25, pp. 12569–12588, Oct. 2021, doi: 10.1007/s00500-020-05462-x.
[56] U. K. Jena, P. K. Das, and M. R. Kabat, "Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment," *J. King Saud Univ.-Comput. Inf. Sci.*, early access. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157819309267, doi: 10.1016/j.jksuci.2020.01.012.
[57] T. Thein, M. M. Myo, S. Parvin, and A. Gawanmeh, "Reinforcement learning based methodology for energy-efficient resource allocation in cloud data centers," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 32, no. 10, pp. 1127–1139, Dec. 2020, doi: 10.1016/j.jksuci.2018.11.005.
[58] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "Analysis and lessons from a publicly available Google cluster trace," EECS Dep., RAD Lab, Univ. California Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2010-95, Jun. 2010.
[59] R. T. Marler and J. S. Arora, "The weighted sum method for multi-objective optimization: New insights," *Struct. Multidisciplinary Optim.*, vol. 41, no. 6, pp. 853–862, Jun. 2010, doi: 10.1007/s00158-009-0460-7.
[60] I. Y. Kim and O. L. de Weck, "Adaptive weighted-sum method for bi-objective optimization: Pareto front generation," *Struct. Multidisciplinary Optim.*, vol. 29, no. 2, pp. 149–158, 2005, doi: 10.1007/s00158-004-0465-1.
[61] A. Abdelsamea, A. A. El-Moursy, E. E. Hemayed, and H. Eldeeb, "Virtual machine consolidation enhancement using hybrid regression algorithms," *Egyptian Inform. J.*, vol. 18, no. 3, pp. 161–170, Nov. 2017, doi: 10.1016/j.eij.2016.12.002.
[62] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[63] S. Fairee, S. Prom-On, and B. Sirinaovakul, "Reinforcement learning for solution updating in artificial bee colony," *PLoS ONE*, vol. 13, no. 7, Jul. 2018, Art. no. e0200738, doi: 10.1371/journal.pone.0200738.
[64] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm," *J. Global Optim.*, vol. 39, pp. 459–471, Nov. 2007, doi: 10.1007/s10898-007-9149-x.
[65] R. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23–50, 2011, doi: 10.1002/spe.995.
[66] A. Hussain and M. Aleem, "GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures," *Data*, vol. 3, no. 4, p. 38, 2018, doi: 10.3390/data3040038.
[67] H. Saleh, H. Nashaat, W. Saber, and H. M. Harb, "IPSO task scheduling algorithm for large scale data in cloud computing environment," *IEEE Access*, vol. 7, pp. 5412–5420, 2018.

BOONHATAI KRUEKAEW received the B.Sc. degree in computer science from the Prince of Songkla University, Hat Yai, Songkhla, Thailand, and the M.Sc. degree in computer science from the King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand, where she is currently pursuing the Ph.D. degree. Her research interests include cloud computing, algorithm, artificial intelligence, and swarm intelligence.

WARANGKHANA KIMPAN (Member, IEEE) received the Ph.D. degree in system information engineering from Kagoshima University, Japan. She is currently an Assistant Professor with the Department of Computer Science, School of Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand. Her main research interests include swarm intelligence, biomedical engineering, big data, data science and analytics, cloud computing, and the Internet of Things.

ประวัติผู้เขียน

ชื่อ	นางสาวบุญหทัย เครือแก้ว
วัน เดือน ปีเกิด	29 พฤศจิกายน พ.ศ.2528
ที่อยู่ปัจจุบัน	72/124 คอนโดลุมพินีวิลล์ อ่อนนุช-ลาดกระบัง ถ.ลาดกระบัง เขตลาดกระบัง แขวงลาดกระบัง กรุงเทพฯ 10520
ประวัติการศึกษา	(2550) วิทยาศาสตรบัณฑิต สาขาวิทยาการคอมพิวเตอร์ เกรดเฉลี่ย 3.20 (มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตหาดใหญ่ สงขลา) (2557) วิทยาศาสตรมหาบัณฑิต สาขาวิทยาการคอมพิวเตอร์ เกรดเฉลี่ย 3.83 (สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง กรุงเทพฯ)
ทุนการศึกษาที่ได้รับ	คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ผลงานทางวิชาการ	1. W. Kimpan, B. Kruekaew, Heuristic task scheduling with artificial bee colony algorithm for virtual machines, in Proceeding of 2016 Joint 8th International Conference on Soft Computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS), IEEE, Sapporo, Japan, 2016, pp. 281–286. (Conference Publication) 2. B. Kruekaew, and W. Kimpan, “Enhancing of Artificial Bee Colony Algorithm for Virtual Machine Scheduling and Load Balancing Problem in Cloud Computing,” *Int. J. Comput. Intell. Syst.*, vol. 13, no. 1, pp. 496-510, 2020, doi: 10.2991/ijcis.d.200410.002. (Journal Publication) 3. B. Kruekaew, and W. Kimpan, “Multi-Objective Task Scheduling Optimization for Load Balancing in Cloud Computing Environment Using Hybrid Artificial Bee Colony Algorithm With Reinforcement Learning,” *IEEE ACCESS*, vol. 10, pp. 17803-17818, 2022.