**الجمهـوريـة العربيــة الســـورية**

**المعهد العالي للعلوم التطبيقية والتكنولوجيا**

**قسـم المعلومايتة**

**العام الدراسي 2024/2025**

**مشروع تخرج**

أعد لنيل درجة الإجازة في هندسة البرمجيات والذكاء الصنعي

جدولة مهام في بيئة سحابية باستخدام خوارزمية بطولة الدوري متعددة الأهداف

تقديم

**غفار حيدر**

إشراف

**ما. عماد قرحيلي**

20/7/2025

أهدي هذا العمل إلى .

ضع حكمتك المفضلة هنا (اختياري) مع ذكر قائلها.

كلمة شكر

أتقدم بالشكر إلى

الخلاصة

تُعد الحوسبة السحابية من أبرز التقنيات الحديثة التي أحدثت تحولًا جذريًا في طريقة معالجة البيانات وتقديم الخدمات الحاسوبية. ومن بين التحديات الرئيسية التي تواجه هذا المجال هي عملية جدولة المهام، والتي تتطلب توزيع الموارد بكفاءة لتحقيق أداء عالٍ وتكلفة منخفضة. يهدف هذا المشروع إلى تطوير وتنفيذ خوارزمية MO-LCA لجدولة المهام في بيئة الحوسبة السحابية. تعتمد الخوارزمية على مفهوم البطولة بين الفرق لتحسين حلول الجدولة من خلال التنافس المتكرر والتعلم من استراتيجيات الخصوم، مع أخذ عدة أهداف في الاعتبار. تتضمن خطوات تنفيذ المشروع بناء بيئة المحاكاة باستخدام CloudSim Plus (بلغة Java) لتوليد سيناريوهات افتراضية من المهام والموارد. بالإضافة إلى تصميم وتنفيذ خوارزمية بطولة الدوري بلغة Python، بحيث تتكامل مع بيئة المحاكاة وتحاكي سلوك الجدولة الذكية. دمج الخوارزمية مع بيئة CloudSim Plus باستخدام broker script وملفات إعداد json. كما تم تطوير واجهة ويب تفاعلية باستخدام React وNode.js لتمكين الباحثين أو المستخدمين من ضبط إعدادات المحاكاة ورؤية النتائج بشكل مرئي. يُتوقع أن تسهم هذه الخوارزمية في تحسين كفاءة الجدولة في أنظمة الحوسبة السحابية، فقد أظهرت تفوقًا على الخوارزميات التقليدية من حيث الأداء وتعدد الأهداف المحققة.

Abstract

Cloud computing is one of the most prominent modern technologies that has brought a fundamental transformation in how data is processed and computing services are delivered. One of the main challenges in this field is task scheduling, which requires efficient resource allocation to achieve high performance and low cost. This project aims to develop and implement the MO-LCA (Multi-Objective League Championship Algorithm) for task scheduling in cloud computing environments. The algorithm is inspired by the concept of league tournaments between teams, aiming to improve scheduling solutions through repeated competition and learning from opponents' strategies, while considering multiple objectives. The project implementation involves building a simulation environment using CloudSim Plus (in Java) to generate virtual scenarios of tasks and resources. In addition, the MO-LCA algorithm is designed and implemented in Python, integrated with the simulation environment to simulate intelligent scheduling behavior. Integration is achieved through the use of a broker script and JSON configuration files to connect the algorithm with CloudSim Plus. Furthermore, an interactive web interface is developed using React and Node.js to enable researchers or users to configure simulation settings and visualize the results. This algorithm is expected to enhance scheduling efficiency in cloud computing systems, as it has demonstrated superior performance compared to traditional algorithms in terms of effectiveness and multi-objective optimization.

المحتويات

[الخلاصة 3](#_Toc204789663)

[المحتويات 5](#_Toc204789664)

[قائمة الأشكال 10](#_Toc204789665)

[قائمة الجداول 11](#_Toc204789666)

[الاختصارات 12](#_Toc204789667)

[الفصل الأول 14](#_Toc204789668)

[1- التعريف بالمشروع 14](#_Toc204789669)

[1.1- مقدمة 15](#_Toc204789670)

[2.1- هدف المشروع 15](#_Toc204789671)

[3.1- متطلبات المشروع 16](#_Toc204789672)

[الفصل الثاني 17](#_Toc204789673)

[2- الدراسة النظرية 17](#_Toc204789674)

[1.2- البيئة السحابية 18](#_Toc204789675)

[1.1.2- نماذج النشر (Deployment Models): 18](#_Toc204789676)

[2.1.2- خصائص البيئة السحابية: 18](#_Toc204789677)

[3.1.2- نماذج الخدمات السحابية 19](#_Toc204789678)

[4.1.2- تحديات الحوسبة السحابية 20](#_Toc204789679)

[2.2- دراسة المصطلحات ذات الصلة بالحوسبة السحابية 21](#_Toc204789680)

[1.2.2- Cloudlets 21](#_Toc204789681)

[2.2.2- Virtual Machines (VMs) - الحواسب الافتراضية 21](#_Toc204789682)

[3.2.2- DatacenterBroker (وسيط مركز البيانات) 22](#_Toc204789683)

[4.2.2- Schedulers (الجدولة) 22](#_Toc204789684)

[3.2- جدولة المهام (Task scheduling): 23](#_Toc204789685)

[1.3.2- تصنيفات تقنيات الجدولة: 24](#_Toc204789686)

[4.2- الخوارزميات التطورية (Evolutionary Algorithms - EAs) 24](#_Toc204789687)

[1.4.2- أنواع الخوارزميات التطورية: 24](#_Toc204789688)

[5.2- خوارزمية بطولة الدوري (League Championship Algorithm - LCA) 25](#_Toc204789689)

[6.2- الأمثلة متعددة الأهداف (Multi-Objective Optimization) 27](#_Toc204789690)

[1.6.2- مفهوم الحلول المثلى (Pareto Optimal Solutions): 27](#_Toc204789691)

[2.6.2- استراتيجيات حل مشكلات الأمثلة متعددة الأهداف: 27](#_Toc204789692)

[الفصل الثالث 29](#_Toc204789693)

[3- دراسة مرجعية 29](#_Toc204789694)

[1.3- تطبيق خوارزمية MO-LCA لتحسين جدولة سلسلة الإمداد في نظام تصنيع متعدد المواقع. 30](#_Toc204789695)

[1.1.3- جدولة الموردين (Supplier Scheduling): 30](#_Toc204789696)

[2.1.3- جدولة المركبات (Vehicle Scheduling): 30](#_Toc204789697)

[3.1.3- الخوارزمية المستخدمة Multiple League Championship Algorithm (MLCA): 30](#_Toc204789698)

[4.1.3- تقنيات التقييم متعددة الأهداف: 32](#_Toc204789699)

[5.1.3- مبدأ Pareto Optimality: 32](#_Toc204789700)

[6.1.3- النتائج: 32](#_Toc204789701)

[2.3- أعمال سابقة في جدولة المهام في بيئة سحابية 32](#_Toc204789702)

[الفصل الرابع 37](#_Toc204789703)

[4- الدراسة التحليلية 37](#_Toc204789704)

[1.4- دراسة المصطلحات ذات الصلة ببطولات الدوري الرياضي 38](#_Toc204789705)

[2.4- خوارزمية MO-LCA 39](#_Toc204789706)

[1.2.4- مخطط دفق الخوارزمية 39](#_Toc204789707)

[2.2.4- جدولة مباريات الدوري league schedule 40](#_Toc204789708)

[3.2.4- دالة الملائمة Fitness function 40](#_Toc204789709)

[4.2.4- تحديد الفائز في المباراة 41](#_Toc204789710)

[5.2.4- تحليل المباريات 42](#_Toc204789711)

[6.2.4- Pseudo Code 46](#_Toc204789712)

[3.4- دوال الهدف objective functions 46](#_Toc204789713)

[1.3.4- وقت التنفيذ (makespan) 47](#_Toc204789714)

[2.3.4- حساب الكلفة (Cost) 47](#_Toc204789715)

[3.3.4- تنفيذ دالة الهدف في MO-LCA 48](#_Toc204789716)

[الفصل الخامس 49](#_Toc204789717)

[5- الأدوات والبيئات المستخدمة 49](#_Toc204789718)

[1.5- بيئة المحاكاة السحابية CloudSim Plus 50](#_Toc204789719)

[2.5- اللغات البرمجية المستخدمة 51](#_Toc204789720)

[1.2.5- Java 51](#_Toc204789721)

[2.2.5- Python 51](#_Toc204789722)

[3.2.5- React 51](#_Toc204789723)

[4.2.5- Node.js 51](#_Toc204789724)

[5.2.5- Javascript 51](#_Toc204789725)

[6.2.5- Git 52](#_Toc204789726)

[7.2.5- GitHub 52](#_Toc204789727)

[الفصل السادس 53](#_Toc204789728)

[6- تصميم النظام 53](#_Toc204789729)

[1.6- هيكلة المشروع 54](#_Toc204789730)

[2.6- مخطط الصفوف Class diagram 56](#_Toc204789731)

[3.6- تنفيذ وحدة الخوارزمية 57](#_Toc204789732)

[4.6- تهيئة بيئة المحاكاة والتكامل مع الخوارزمية 59](#_Toc204789733)

[5.6- واجهة التطبيق 61](#_Toc204789734)

[1.5.6- الواجهة الخلفية (backend): 61](#_Toc204789735)

[2.5.6- الواجهة الأمامية (frontend): 62](#_Toc204789736)

[6.6- واجهة التكامل 67](#_Toc204789737)

[7.6- خطة الاختبارات 67](#_Toc204789738)

[8.6- Logging 68](#_Toc204789739)

[الفصل السابع 69](#_Toc204789740)

[7- تجارب وملاحظات: 69](#_Toc204789741)

[1.7- وضع نماذج الاختبار 70](#_Toc204789742)

[2.7- تجارب أولية: 70](#_Toc204789743)

[1.2.7- تابع كلفة خطي 70](#_Toc204789744)

[2.2.7- تابع كلفة غير خطي 73](#_Toc204789745)

[3.2.7- تابع كلفة يراعي استهلاك الطاقة 78](#_Toc204789746)

[3.7- نتائج المحاكاة 78](#_Toc204789747)

[1.3.7- حل مشكلة الscale 80](#_Toc204789748)

[الفصل السابع 82](#_Toc204789749)

[8- اختبارات اضافية 82](#_Toc204789750)

[المراجع 83](#_Toc204789751)

قائمة الأشكال

[الشكل 1 نماذج الخدمات السحابية 20](#_Toc204789752)

[الشكل 2 نظام جدولة المهام في بيئة الحوسبة السحابية 23](#_Toc204789753)

[الشكل 3 مثال عن هيكل خوارزمية جينية متعددة الأهداف 28](#_Toc204789754)

[الشكل 4 مقارنة حسب مدة التنفيذ 34](#_Toc204789755)

[الشكل 5 مقارنة حسب الكلفة 34](#_Toc204789756)

[الشكل 6 مقارنة توزيع الحمل 36](#_Toc204789757)

[الشكل 7 مقارنة زمن الاستجابة 36](#_Toc204789758)

[الشكل 8 مخطط تدفق خوارزمية MO-LCA 39](#_Toc204789759)

[الشكل 9 مخطط جدولة مباريات الدوري 40](#_Toc204789760)

[الشكل 10 تحليل المباريات 43](#_Toc204789761)

[الشكل 11 مخطط صندوق النظام 55](#_Toc204789762)

[الشكل 12 مخطط الصفوف 56](#_Toc204789763)

[الشكل 13 مثال عن نتائج المحاكاة 60](#_Toc204789764)

[الشكل 14 مثال عن نتائج الكلفة في المحاكاة 60](#_Toc204789765)

قائمة الجداول

[الجدول 1 مطابقة االمصطلحات بين LCA و EA 26](#_Toc204789766)

[الجدول 2 تحسينات من ناحية وقت التنفيذ الكلي 35](#_Toc204789767)

[الجدول 3 تحسينات من نحاية استهلاك الطاقة 35](#_Toc204789768)

[الجدول 4 حسينات من ناحية وقت التشغيل 35](#_Toc204789769)

[الجدول 5 مصفوفة SWOT 44](#_Toc204789770)

[الجدول 6 مقارنة بين أدوات محاكاة السحابة 50](#_Toc204789771)

[الجدول 8 مخطط اختبار سير العمل الكلي للنظام 67](#_Toc204789772)

[الجدول 7 أنماط المسائل 70](#_Toc204789773)

الاختصارات

|  |  |  |
| --- | --- | --- |
| الاختصار | المصطلح بالغة الانكليزية | المصطلح باللغة العربية |
| LCA | League Championship Algorithm | خوارزمية بطولة الدوري |
| MO-LCA | Multiple Objective League Championship Algorithm | خوارزمية بطولة الدوري متعددة الأهداف |
| SaaS | Software as a service | البرمجيات كخدمة |
| Paas | Platform as a service | المنصة التحتية كخدمة |
| Iaas | Infrastructure as a Service | البنية التحتية كخدمة |
| Faas | Functions as a Service | الوظيفة كخدمة |
| BaaS | Backend as a Service | الواجهة الخلفية كخدمة |
| VM | Virtual Machine | آلة افتراضية |
| EA | Evolutionary Algorithm | الخوارزمية التطورية |
| GA | Genetic Algorithm | الخوارزمية الجينية |
| GP | Genetic Programming | البرمجة الجينية |
| EP | Evolutionary Programming | برمجة التطورية |
| ES | Evolution Strategies | استراتيجيات التطور |
| ACO | Ant Colony Optimization | أمثلة مستعمرة النمل |
| PSO | Particle Swarm Optimization | أمثلة سرب الجسيمات |
| SA | Simulated Annealing | محاكاة التلدين |
| CPU | Central Processing Unit | وحدة المعالجة المركزية |
| MIPS | Million Instructions Per Second | ملايين التعليمات في الثانية |
| RAM | Random Access Memory | الذاكرة العشوائية |
| CFS | Completely Fair Scheduler | الجدولة الزمنية العادلة بالكامل |
| JSON | JavaScript Object Notation | ترميز كائنات جافا سكريب |
| pe | processing unit | وحدة معالجة |
|  |  |  |

الفصل الأول

1. التعريف بالمشروع

يتطرّق هذا الفصل إلى أهداف المشروع ومتطلباته مع تعداد للأهداف البحثية المراد إجراءها واللازمة لتكوين فهم عميق للمشروع.

* 1. مقدمة

تُعد الحوسبة السحابية (Cloud Computing) من أبرز تقنيات العصر الرقمي، حيث تتيح للمستخدمين الوصول إلى موارد حاسوبية افتراضية غير محدودة حسب الطلب، ووفق نماذج تسعير مرنة. فمن أبرز وأوسع استخدامات الحوسبة السحابية اليوم: خدمات تحليل البيانات الضخمة (Big Data)، تخزين الملفات ومشاركتها، نماذج اللغة الكبيرة (LLMs) مثل ChatGPT وGoogle Gemini، وخدمات توليد الصور والفيديو بالذكاء الاصطناعي مثل DALL·E وMidjourney وغيرها من الخدمات المتنوعة. تتطلب هذه الأنظمة قدرًا هائلًا من القدرة الحسابية والذاكرة ومعالجة البيانات، وهو ما توفره الحوسبة السحابية بشكل مرن وقابل للتوسع، مما يجعل هذه الخدمات ممكنة وفعّالة على نطاق واسع. وتُستخدم هذه التقنيات في مجالات مثل التعليم، والصحة، وصناعة المحتوى، وخدمة العملاء، والبرمجة التوليدية.

ومع ذلك، يواجه هذا المجال تحديات جوهرية، من أبرزها جدولة المهام، والتي تتمثل في عملية توزيع المهام المقدمة من المستخدمين على الأجهزة الافتراضية المتاحة، مع مراعاة معايير جودة الخدمة (QoS) مثل تقليل زمن التنفيذ (Makespan)، وتحسين استخدام الموارد، وخفض التكلفة.

ونظرًا لتعقيد مشكلة جدولة الموارد في بيئة الحوسبة السحابية، والتي تُصنّف ضمن مشاكل NP-Hard، فإن الأساليب التقليدية (الحتمية) غالبًا ما تعجز عن تقديم حلول فعالة أو قريبة من المثلى خلال زمن معقول. لذلك، اتجهت الأبحاث الحديثة نحو استخدام الخوارزميات التطورية والذكية لمعالجة هذه التحديات، حيث تمتاز بقدرتها على التكيف مع تغيّر الأحمال وتحقيق أهداف متعددة ومتضاربة أحيانًا.

استجابة لهذه التحديات، يقترح هذا المشروع خوارزمية ذكية جديدة تُعرف بـخوارزمية بطولة الدوري متعددة الأهداف، والمستوحاة من مفهوم بطولات الدوري الرياضية، حيث تتنافس "الفرق" (الحلول) في جولات متعددة وتتعلم من استراتيجيات الخصوم لتحسين أدائها. تهدف الخوارزمية إلى تحسين عملية جدولة المهام من خلال موازنة عدة أهداف، مثل تقليل زمن التنفيذ، وتحقيق توزيع عادل للحمل، وتقليل استهلاك الطاقة.

تُقدم هذه الخوارزمية نموذجًا ذكيًا ومرنًا لمعالجة تحديات جدولة المهام في بيئة الحوسبة السحابية، وقد أظهرت نتائج أولية تفوقها على الطرق التقليدية من حيث الكفاءة وتحقيق أهداف متعددة، مما يعزز من موثوقية وفعالية البنية التحتية السحابية الحديثة.

* 1. هدف المشروع

يهدف هذا المشروع إلى تطوير خوارزمية Multiple Objective League Championship Algorithm (اختصاراً – MO-LCA) واستخدامها في جدولة المهام في البيئة السحابية.

* 1. متطلبات المشروع

المتطلبات البحثية

* إجراء دراسة حول مفاهيم الخوارزميات التطورية عموماً ومتعددة الأهداف خصوصاً.
* اجراء دراسة مرجعية كافية حول خوارزميات جدولة المهام.
* التعرّف على مفهوم السحابة IaaS.
* مقارنة بين أدوات محاكاة السحابة الالكترونية.

المتطلبات الوظيفية

* بناء محاكي لسحابة الكترونية يسمح بإعداد الموارد اختياريًّا.
* السماح بإنشاء مجموعة من المهام بأزمنة تنفيذ مختلفة (مع خيار توليدها عشوائيًّا).
* إعداد مجدول مهام يقوم بإسناد المهام إلى موارد السحابة وفق الخوارزمية المقترحة.
* إعادة تنفيذ الجدولة باستخدام خوارزميات شهيرة على نفس مجموعات المهام
* تقييم نتائج الجدولة باستخدام معايير التقييم (وقت التنفيذ، الكلفة، وقت التشغيل)
* توليد مخططات بيانية إحصائية توضح أداء الخوارزمية المقترحة مع أداء الخوارزميات الأخرى.

المتطلبات غير الوظيفية

* سهولة الاستخدام: يجب أن يستطيع المستخدم إجراء الجدولة بأقل جهد ممكن.
* قابلية التوسّع: إمكانية إضافة خوارزميات أخرى للمجدول بشكل مرن، مع إمكانية توسعة الخوارزمية دون الحاجة إلى إعادة برمجة من نقطة الصفر.
* الأداء: يجب تتناسب سرعة الجدولة مع عدد المهام بشكل نسبي (ألا يتجاوز الدقيقة الواحدة لـ 10^4 مهمة كمثال)

الفصل الثاني

1. الدراسة النظرية

يعرض هذا الفصل أهم المفاهيم النظرية اللازم فهمها لإنجاز المشروع بما فيها إجراء دراسات مقارنة مع أعمال مشابهة كدراسة مرجعية قبل البدء بوضع حل خوارزمي وفق المنهج المقترح.

* 1. البيئة السحابية

هي نموذج يتيح الوصول الشامل والمريح إلى مجموعة مشتركة من موارد الحوسبة القابلة للتكوين (مثل الشبكات، الخوادم، التخزين، التطبيقات، والخدمات)، والتي يمكن توفيرها وإطلاقها بسرعة مع جهد إداري ضئيل أو تفاعل محدود مع مزود الخدمة.

* + 1. نماذج النشر (Deployment Models):

تتوفر عدة نماذج للسحابة هي:

السحابة العامة (Public Cloud): خدمات متاحة للجميع عبر الإنترنت (AWS, Azure).

السحابة الخاصة (Private Cloud): مخصصة لمؤسسة واحدة، مع مزيد من التحكم والأمان.

السحابة الهجينة (Hybrid Cloud): دمج بين العامة والخاصة للاستفادة من مرونة وأمان متوازن.

السحابة المجتمعية (Community Cloud): مخصصة لمجموعة مؤسسات لها احتياجات مشتركة. ‎[2]

* + 1. خصائص البيئة السحابية:

تتلخّص أهم خصائص البيئة السحابية فيما يلي:

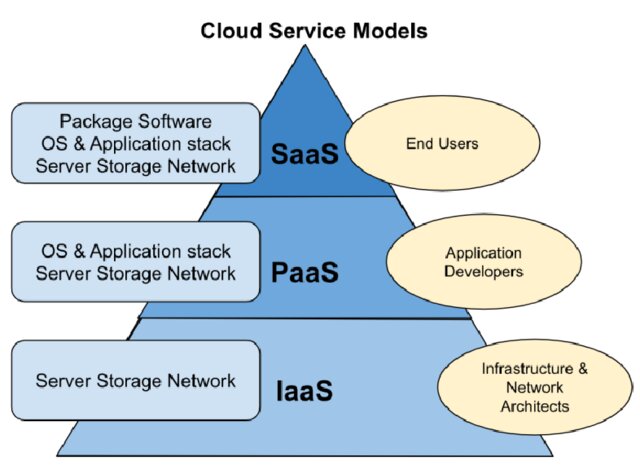
* **الخدمة الذاتية عند الطلب (On-demand self-service):** يمكن للمستخدم توفير إمكانيات الحوسبة، مثل وقت الخادم أو التخزين الشبكي، بشكل تلقائي وفقًا لاحتياجاته دون الحاجة إلى تدخل بشري من مزود الخدمة في كل مرة.
* **الوصول الواسع عبر الشبكة (Broad network access):** تتوفر الإمكانيات عبر الشبكة ويمكن الوصول إليها من خلال منصات مختلفة ومتنوعة (مثل الهواتف المحمولة، الأجهزة اللوحية، أجهزة الكمبيوتر المحمولة، ومحطات العمل).
* **تجميع الموارد (Resource pooling):** يتم تجميع موارد الحوسبة الخاصة بمزود الخدمة لخدمة عدة مستخدمين باستخدام نموذج متعدد المستأجرين (Multi-tenant model)، حيث يتم تخصيص الموارد المادية والافتراضية وإعادة تخصيصها ديناميكيًا بناءً على طلب المستخدمين.
* **المرونة السريعة (Rapid elasticity):** يمكن توفير القدرات (الموارد المختلفة) أو تحريرها بمرونة، وفي بعض الحالات يتم ذلك تلقائيًا، لتوسيع أو تقليص الموارد بسرعة حسب الطلب. بالنسبة للمستخدم، تظهر القدرات وكأنها غير محدودة ويمكن الحصول عليها بأي كمية وفي أي وقت.
* **الخدمة المقاسة (Measured service):** تقوم الأنظمة السحابية تلقائيًا بمراقبة واستخدام الموارد بشكل محكم من خلال قدرات قياس مناسبة لنوع الخدمة (مثل التخزين، المعالجة، عرض النطاق الترددي، وعدد الحسابات النشطة). يمكن مراقبة الموارد والتحكم فيها والإبلاغ عنها، مما يضمن الشفافية لكل من المزود والمستخدم فيما يتعلق بالخدمات المستهلكة.‎[2]
  + 1. نماذج الخدمات السحابية

يوجد ثلاث نماذج من الخدمات السحابية تستخدم على نطاق واسع في الشركات وهي:

* البرمجيات كخدمة (Software as a Service – SaaS):تمكن هذه الخدمة المستخدم من الوصول لتطبيقات مزود الخدمة واستخدامها من خلال متصفحات الويب أو واجهات برمجية. مثل Zoom, Netflix, Gmail.
* المنصة كخدمة (Platform as a Service – PaaS):تمكن هذه الخدمة المستخدم من نشر تطبيقات قام بتطويرها أو الحصول عليها باستخدام البيئات والمكتبات التي يدعمها مزود الخدمة. أمثلة عن هذه الخدمة AWS Elastic Beanstalk, Google App Engine.
* البنية التحتية كخدمة (Infrastructure as a Service – IaaS):هذه الخدمة هي واحدة من الطبقات الرئيسية للحوسبة السحابية حيث توفر للمستخدم القدرة على **تخصيص موارد المعالجة، التخزين، الشبكات، وغيرها من الموارد الحاسوبية الأساسية،** مما يسمح له بنشر وتشغيل البرمجيات المختلفة. أهم الأمثلة عنها Amazon Web Services, Microsoft Azure, and Google Compute Engine. ‎[1]

كما توجد نماذج أقل شهرة مثل:

* الوظيفة كخدمة (Functions as a Service – FaaS): مثل AWS Lambda، لتنفيذ وظائف دون إدارة الخوادم.
* الواجهة الخلفية كخدمة (Backend as a Service – BaaS):مثل Firebase، لتقديم خدمات جاهزة للواجهة الخلفية.



الشكل 1 نماذج الخدمات السحابية

* + 1. تحديات الحوسبة السحابية
* **أمان البيانات والخصوصية**: تُعد أمان البيانات وخصوصيتها من أبرز التحديات في الحوسبة السحابية، حيث يتم تخزين بيانات حساسة ومهمة على بنية تحتية سحابية خارج سيطرة المؤسسة. قد تتعرض البيانات للاختراق أو السرقة، مما يؤثر على سمعة الشركة ويؤدي إلى خسائر مالية. كما أن تبادل البيانات الضخم عبر الشبكة يعرضها لاحتمالات التسريب إذا لم تكن محمية بشكل كافٍ.
* **إدارة التكاليف**: على الرغم من أن نموذج "الدفع حسب الاستخدام" (Pay As You Go) يساهم في تقليل التكاليف الإجمالية، إلا أن هناك بعض التحديات في إدارة الموارد بكفاءة. عدم تحسين استخدام الخوادم أو ترك الخدمات السحابية مفعلة دون حاجة، مثل نسيان إيقاف تشغيل خادم افتراضي أثناء عطلة نهاية الأسبوع، يمكن أن يرفع التكاليف دون فائدة.
* **بيئات السحابة المتعددة (Multi-Cloud Environments)**: تتجه العديد من المؤسسات إلى استخدام أكثر من مزود خدمة سحابية لتحقيق الاستفادة القصوى من الخدمات المختلفة. ومع ذلك، فإن إدارة هذه البيئات تصبح معقدة للغاية بالنسبة لفرق تقنية المعلومات نظرًا لاختلافات البنى التحتية بين مقدمي الخدمات، مما يصعّب عملية التكامل والإدارة.
* **تحديات الأداء**: يؤثر أداء الخدمات السحابية بشكل مباشر على رضا المستخدمين؛ أي تأخير بسيط في تحميل التطبيقات أو صفحات الويب يمكن أن يؤدي إلى انخفاض في عدد المستخدمين. من أسباب ذلك ضعف توزيع الأحمال (Load Balancing) على الخوادم وعدم كفاءة تخصيص الموارد، مما يقلل من كفاءة التجربة للمستخدمين.
* **التوافقية والمرونة (Interoperability and Flexibility)**: عند الرغبة في الانتقال من مزود سحابي إلى آخر، غالبًا ما تواجه المؤسسات صعوبات كبيرة. ذلك لأن التطبيقات التي تم تطويرها على منصة معينة قد تحتاج إلى إعادة كتابة كاملة لتعمل على منصة أخرى.
* **الاعتماد العالي على الشبكة**: تعتمد الحوسبة السحابية بشكل كامل على الشبكة عالية السرعة لتوفير الموارد في الوقت الحقيقي. في حال حدوث انقطاع في الإنترنت أو ضعف في سرعة الاتصال، قد تتأثر الخدمات بشكل كبير، مما يؤدي إلى خسائر في الأعمال.
  1. دراسة المصطلحات ذات الصلة بالحوسبة السحابية

نشرح في هذا القسم أهم المصطلحات النظرية والعملية التي تستخدم في الحوسبة السحابية.

* + 1. Cloudlets

تمثل المهام الفردية أو الوظائف التي يتم إرسالها إلى البيئة السحابية.

لكل مهمة خصائص معينة فريدة:

* الطول (Length): عدد التعليمات المطلوب تنفيذها.
* حجم الملف (File Size).
* عدد وحدات المعالجة المركزية المطلوبة (CPUs Needed).
* الأولوية (Priority).

أثناء المحاكاة يتم تنفيذ هذه المهام على الحواسب الافتراضية (VMs)، والتي تستضيفها الحواسب الفيزيائية (Hosts).

* + 1. Virtual Machines (VMs) - الحواسب الافتراضية

الحواسب الافتراضية هي وحدات الحوسبة التي تُنفذ فيها المهام (Cloudlets).

يتم تجهيز الحواسب الافتراضية بموارد مثل:

* ملايين التعليمات في الثانية (MIPS).
* الذاكرة العشوائية (RAM).
* عرض النطاق الترددي (Bandwidth).
* التخزين (Storage).

يتم محاكاة الحواسب الافتراضية على حواسب فيزيائية أو مضيفين (Hosts).

* + 1. DatacenterBroker (وسيط مركز البيانات)

وسيط مركز البيانات مسؤول عن:

* **تخطيط وجدولة** المهام على الآلات الافتراضية.
* يعتبر من أهم الكيانات في أدوات محاكاة CloudSim، حيث لا يمكن تنفيذ عمليات التوفير والجدولة بدونه.
* بعد تهيئة المهام والحواسب الافتراضية في المحاكاة، يتم إرسالها إلى **DatacenterBroker**، حيث يتولى مسؤولية إدارة توزيع المهام على الحواسب الافتراضية بناءً على سياسة الجدولة المعتمدة، ومراقبة تنفيذ المهام.
  + 1. Schedulers (الجدولة)

تؤثر الجدولة في كيفية تنفيذ المهام داخل الحواسب الافتراضية أثناء عملية جدولة المهام في CloudSim.

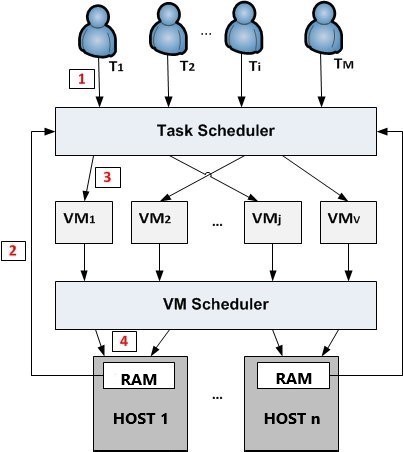
توفر Cloudsim-plus ثلاثة أنواع رئيسية للجدولة:

* الجدولة المشتركة بالوقت (Time-Shared Scheduler): في هذا النوع، تشترك المهام في موارد الحاسب الافتراضي، ويتم تنفيذ المهام في شرائح زمنية.
* هذا النهج يحاكي خوارزمية Round Robin.
* الجدولة المشتركة بالمكان (Space-Shared Scheduler): في هذا النوع، تحصل كل مهمة على وصول كامل إلى موارد الحاسب الافتراضي لفترة زمنية محددة. يجب على المهام الأخرى الانتظار حتى يكتمل تنفيذ المهمة الحالية. هذه الخوارزمية تحاكي نظام First Come First Serve.
* الجدولة الزمنية العادلة بالكامل (Completely Fair Scheduler - CFS): يحاكي هذا النوع الجدولة المستخدمة في نواة لينوكس وتهدف الى استخدام موارد ال (CPU) الى اقصى حد ممكن.
  1. جدولة المهام (Task scheduling):

تُعد عملية جدولة المهام في بيئات الحوسبة السحابية من المهام الحيوية لضمان تحقيق الأداء المطلوب للخوادم المستخدمة. يتم في هذه البيئات استخدام مجموعة متنوعة من الخوادم التي قد تكون موزعة جغرافيًا وتعمل معًا لتوفير مجموعة من الموارد للحوسبة السحابية المتكاملة.

تنشأ في هذه العملية العديد من التحديات والقضايا، مثل اختلاف مستويات الأمان، والتفاوت في كفاءة الأداء، وضمان الوثوقية، ودعم القدرة على التوسع مع تحمل الأخطاء. وتظهر هذه التحديات بوضوح عند وجود عدد كبير من الطلبات الموجهة إلى مورد واحد، مما يؤدي إلى زيادة التحميل عليه مقارنةً بالموارد الأخرى، وبالتالي قد يحدث اختناق في الأداء.

لتجنب هذه المشاكل، يعتبر التخصيص الأمثل للموارد أمرًا ضروريًا، لتحقيق أفضل أداء ممكن وتقليل احتمالية زيادة الحمل على مورد معين، مما يساهم في تحسين الإنتاجية، خاصة في البيئات غير المتجانسة.



الشكل 2 نظام جدولة المهام في بيئة الحوسبة السحابية

يوضح الشكل 2 نظام جدولة المهام في بيئة الحوسبة السحابية: حيث يقدم عدة مستخدمين عددا من المهام المطلوب تنفيذها، يقوم مجدول المهام (Task Scheduler) بتخصيص كل مهمة الى حاسب افتراضي (Virtual Machine)، الحاسب الافتراضي الواحد قد يخصص له أكثر من مهمة، بينما يقوم مجدول الحواسب ا الافتراضية (VM Scheduler) بتوزيع الحواسيب الافتراضية على الموارد الفيزيائية المتوافرة.

* + 1. تصنيفات تقنيات الجدولة:

يمكن تصنيف تقنيات جدولة المهام وفقًا لعدة معايير، أبرزها:

1. **Static Scheduling vs Dynamic Scheduling:**

* **الجدولة الثابتة (Static Scheduling)** يتم اتخاذ جميع قرارات الجدولة قبل بدء التنفيذ، بناءً على معلومات معروفة مسبقًا عن النظام والمهام.
* **الجدولة الديناميكية (Dynamic Scheduling)** يتم اتخاذ القرارات أثناء وقت التشغيل، حيث تُعدل التوزيعات بناءً على التغيرات في الأحمال وحالة الموارد.

1. **الجدولة حسب الأولوية (Priority-based Scheduling):**

يتم تخصيص الموارد وفقًا لمستوى أولوية كل مهمة، بحيث تحصل المهام الحرجة على أولوية أعلى.

1. **الجدولة التقديرية Heuristic-based Scheduling:**

تُستخدم خوارزميات تقريبية (مثل Genetic Algorithms، PSO) لإيجاد حلول فعّالة عندما يصعب الوصول إلى الحلول المثلى.

* 1. الخوارزميات التطورية (Evolutionary Algorithms - EAs)

تمثل الخوارزميات التطورية (EAs) إحدى التقنيات الحسابية المستوحاة من مبادئ التطور الطبيعي. تُستخدم هذه الخوارزميات لحل مشاكل الأمثلة المعقدة، حيث تحاكي آليات التطور البيولوجي مثل: **الاستنساخ، الطفرة، إعادة التركيب، والانتقاء الطبيعي**.

في هذا السياق، تُعتبر الحلول المرشحة بمثابة أفراد في مجموعة سكانية، ويقاس أداؤها من خلال **دالة الملائمة (Fitness Function)** التي تحدد مدى جودة الحل في معالجة المشكلة.

* + 1. أنواع الخوارزميات التطورية:
* **الخوارزميات الجينية (Genetic Algorithms - GA):**

تُعد الخوارزميات الجينية من أكثر الخوارزميات التطورية شيوعًا، حيث تعتمد على محاكاة عملية التطور الطبيعي لإيجاد الحلول المثلى. يتم تمثيل الحلول كسلاسل رقمية تُشبه الكروموسومات.

* **البرمجة الجينية (Genetic Programming - GP):**

تختلف البرمجة الجينية عن الخوارزميات الجينية في طريقة تمثيل الحلول؛ إذ يتم تمثيلها على شكل برامج حاسوبية. يتم تقييم هذه البرامج بناءً على قدرتها على حل المشكلات المحددة، مع إمكانية تطوير تركيبة البرنامج عبر عمليات شبيهة بالتطور الطبيعي.

* **البرمجة التطورية (Evolutionary Programming - EP):**

تشترك البرمجة التطورية مع البرمجة الجينية في الفلسفة العامة، لكنها تتميز بأن هيكل البرنامج يكون ثابتًا، بينما تتركز عملية التطوير في تعديل المعاملات العددية الخاصة بالبرنامج. تُستخدم **EP** عادةً في التطبيقات التي تتطلب تحسين معاملات رياضية دون تغيير هيكلية البرنامج نفسه.

* **استراتيجيات التطور (Evolution Strategies - ES):**

تركّز استراتيجيات التطور على تمثيل الحلول كمُتجهات من الأرقام الحقيقية، وتُعتبر مناسبة للتطبيقات التي تتطلب أمثلة دقيقة على القيم العددية. تتميز هذه الخوارزميات باستخدام تقنيات **التكيف الذاتي (Self-adaptation)**، التي تُمكّنها من تعديل معدلات الطفرة تلقائيًا خلال عملية البحث للوصول إلى حلول أكثر دقة وكفاءة. ‎[3]

* 1. خوارزمية بطولة الدوري (League Championship Algorithm - LCA)

تُعد **خوارزمية بطولة الدوري (LCA)** إحدى الخوارزميات التطورية الحديثة، والمستوحاة من آلية التنافس في البطولات الرياضية. تحاكي هذه الخوارزمية بيئة دوري رياضي افتراضي حيث تتنافس عدة فرق افتراضية لعدة أسابيع (تكرارات). بناءً على جدول الدوري الأسبوعي.

يعتمد إطار عمل الخوارزمية على تعداد السكان العشوائي من أجل الأمثلة العالمية (Global Optimization) على مساحة بحث مستمرة. السمة المشتركة بين جميع الخوارزميات القائمة على السكان مثل LCA هي أنها تحاول نقل مجموعة من الحلول الممكنة إلى مناطق واعدة من مساحة البحث أثناء البحث عن الحل الأمثل. تشكل مجموعة حلول يتم اختيارها بشكل عشوائي من مساحة البحث تعداد السكان الاولي للخوارزمية. وباستخدام المصطلحات الرياضية، فإن الدوري في خوارزمية LCA تعني مجموعة السكان، الفريق يعني الحل، والأسبوع يعني التكرار. وكما في معظم الخوارزميات القائمة على تعداد السكان، فإنه في خوارزمية LCA يتطور السكان خلال التكرارات المتتالية، مع الحفاظ على حجم مجموعة السكان ثابتاً.

وفي الجدول 1 مطابقة بسيطة بين مصطلحات خوارزمية LCA و EA.

|  |  |
| --- | --- |
| LCA | EA |
| League | Population |
| Week | Iteration |
| Team | member in the population |
| Team formation | Solution |
| Playing strength | Fitness value |
| Number of seasons | Maximum iterations |

الجدول 1 مطابقة االمصطلحات بين LCA و EA

يرتبط كل حل في مجموعة السكان بأحد الفرق من الدوري وهو التشكيل الحالي للفريق، ولكل حل قيمة لياقة معينة تقيس درجة تكيفه مع الهدف المنشود (أو عدة أهداف)، يمكن تفسيرها على أنها قوة اللعب لهذا الفريق وترتبط بتشكيل الفريق.

في جميع الخوارزميات القائمة على السكان تقريبًا يتم تطبيق سلسلة متعاقبة من العوامل على الأفراد في كل تكرار لتوليد الحلول الجديدة للتكرار التالي، الهدف من تطبيق هذه العوامل هو جعل الافراد أكثر ملائمة. وفي خوارزمية LCA يتم الاستفادة من عملية محاكاة تحليل المباريات التي يتبعها المدربون في إنشاء تشكيل أفضل للفريق لجعل فوة اللعب أفضل ما يمكن للفوز في المباريات اللاحقة. وهذا يطابق عملية الاختيار (selection) في الخوارزميات التطورية حيث يختار المدربون أفضل تشكيل ممكن للفريق للصول على أفضل قوة لعب. وتنتهي الخوارزمية بعد عدد معين من المواسم ويتألف كل موسم من أسبوع، ينتج عنه أسابيع من المباريات.

يجب اختيار تمثيل للأفراد في الخوارزميات التطورية، في حالة خوارزمية LCA يمكن تمثيل تشكيل الفريق (الحل) بمتجه بحجم ( يعبر عن عدد متغيرات أو معاملات المشكلة) من الأعداد الحقيقية. يرتبط كل عنصر من هذا المتجه بأحد اللاعبين ويمثل قيمة المتغير المقابل للمشكلة. والتغيير في قيمة هذا العنصر يعبر عن تغيير وظيفة اللاعب في التشكيل الجديد.

نعرف على أنها دالة عددية ب متغير والتي يجب تصغيرها على فضاء القرار.

يمكن تمثيل تشكيل الفريق في الأسبوع بالمتجه ، وتكون هي قيمة الملائمة أو قوة الفريق للتشكيل . ونعرف على أنه أفضل تشكيل للفريق تم اختياره حتى الأسبوع ، ولتحديد نستخدم اختياراً جشعاً استناداً إلى قيمة قوتي اللعب و . ‎[4]

* 1. الأمثلة متعددة الأهداف (Multi-Objective Optimization)

الأمثلة متعددة الأهداف هي أحد مجالات صنع القرار، تهتم بمشكلات الأمثلة الرياضية التي تتضمن أكثر من دالة هدف واحدة تحتاج إلى تحسينها في وقت واحد.

تعتبر الأمثلة متعدد الأهداف نوعًا من الأمثلة المتجهية (Vector Optimization) وقد تم تطبيقها في العديد من المجالات العلمية، بما في ذلك الهندسة، والاقتصاد، والخدمات اللوجستية، حيث يكون من الضروري اتخاذ قرارات مثلى في ظل وجود مقايضات بين هدفين أو أكثر متعارضين. على سبيل المثال، تقليل التكلفة مع زيادة مستوى الراحة عند شراء سيارة، أو تحسين الأداء مع تقليل استهلاك الوقود وانبعاثات الملوثات في المركبات.

* + 1. مفهوم الحلول المثلى (Pareto Optimal Solutions):

في مشكلات الأمثلة متعددة الأهداف، ليس من الضروري أن يوجد حل واحد يستطيع تحسين جميع الأهداف في الوقت نفسه، حيث تُعتبر دوال الأهداف متعارضة. يُطلق على الحل الذي لا يمكن تحسين أي من أهدافه دون التأثير سلبًا على هدف آخر اسم غير مُهيمن (Nondominated)، أو أمثلية باريتو (Pareto Optimal)، أو كفاءة باريتو (Pareto Efficient)، أو غير أدنى (Noninferior).

في هذه الحالة، من الممكن أن يوجد عدد (قد يكون لانهائيًا) من الحلول المثلى حسب مفهوم باريتو، وتُعتبر جميعها متساوية في الجودة ما لم تتوفر معلومات تفضيلية إضافية تحدد الأولويات.

* + 1. استراتيجيات حل مشكلات الأمثلة متعددة الأهداف:

يقوم الباحثون بدراسة مشكلات الأمثلة متعددة الأهداف من زوايا مختلفة، مما أدى إلى ظهور استراتيجيات متنوعة لحلها. تختلف هذه الاستراتيجيات بناءً على الهدف المراد تحقيقه، ويمكن تلخيصها في النقاط التالية:

* إيجاد مجموعة من الحلول المثلى لباريتو (Pareto Optimal Solutions):

الهدف هنا هو العثور على مجموعة حلول تمثل أفضل النتائج الممكنة بحيث لا يمكن تحسين أي هدف دون التأثير سلبًا على هدف آخر. هذه المجموعة تعطي صانع القرار خيارات متعددة يمكن المفاضلة بينها.

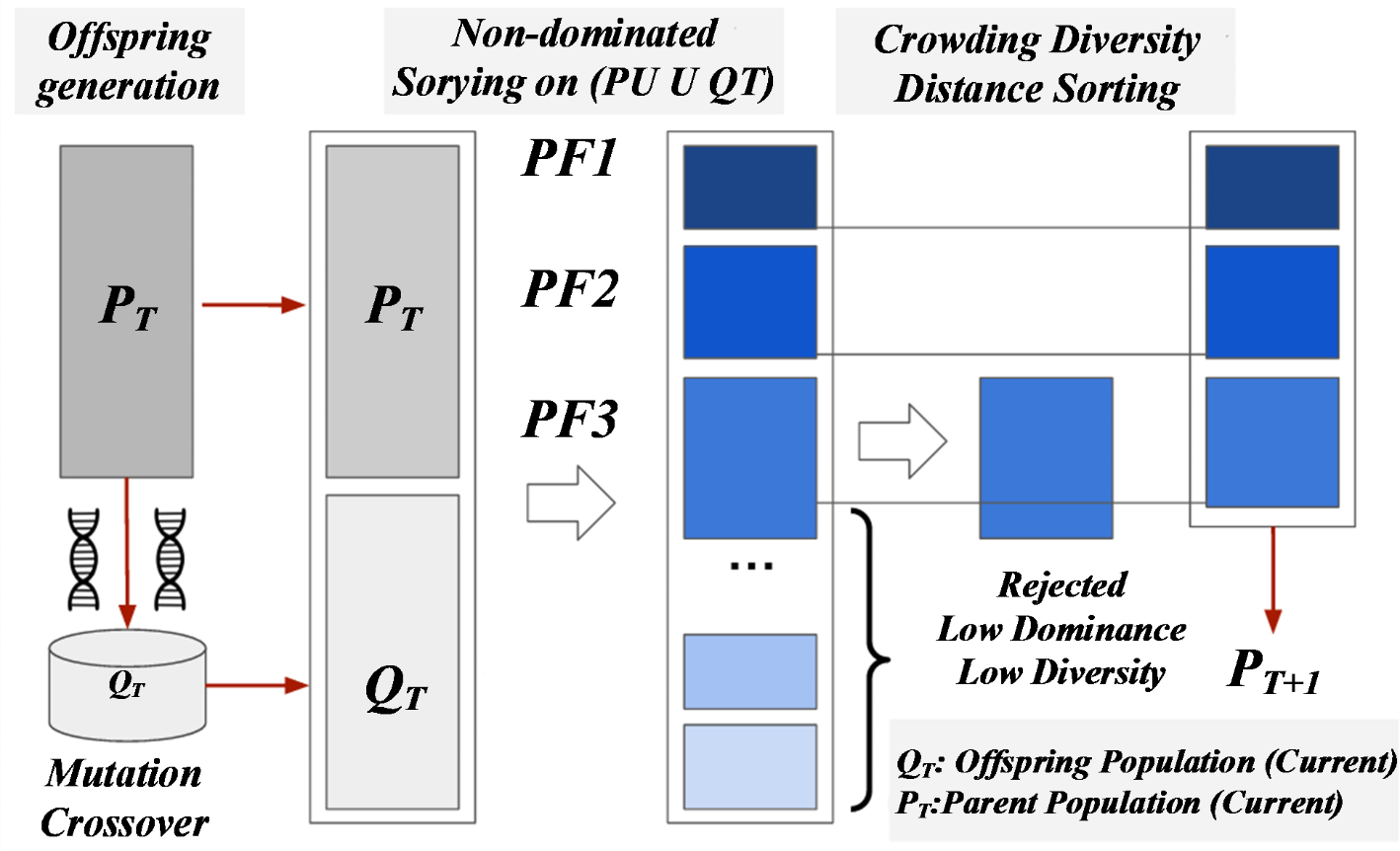
* قياس المقايضات بين الأهداف المختلفة (Quantifying Trade-offs):

في كثير من الأحيان، تحسين أحد الأهداف يتطلب التضحية بجزء من الأهداف الأخرى. تساعد هذه الاستراتيجية على توضيح هذه المقايضات بشكل كمي، مما يمكّن صانع القرار من فهم التأثيرات المترتبة على كل خيار.

* إيجاد حل مفرد يلبي تفضيلات صانع القرار (Single Solution for Decision Maker):

في بعض الحالات، قد يفضل صانع القرار الحصول على حل واحد فقط يلبي تفضيلاته الخاصة، بدلاً من مجموعة كبيرة من الخيارات. في هذه الحالة، يتم تطبيق معايير إضافية لتضييق نطاق الحلول واختيار الأنسب منها. ‎[5]

يوضح الشكل 3 مثال عن هيكل خوارزمية جينية متعددة الأهداف:



الشكل 3 مثال عن هيكل خوارزمية جينية متعددة الأهداف

الفصل الثالث

1. دراسة مرجعية

نستعرض في هذه الفصل بعض الأعمال التي تستخدم خوارزمية MO-LCA وأخرى تهتم بالجدولة السحابية.

* 1. تطبيق خوارزمية MO-LCA لتحسين جدولة سلسلة الإمداد في نظام تصنيع متعدد المواقع.

قام الباحثون في هذا العمل بتطوير نموذج متكامل لجدولة سلاسل الإمداد متعددة المواقع باستخدام خوارزمية Multiple League Championship Algorithm (MLCA) متعددة الأهداف. هذه الخوارزمية هي امتداد للخوارزمية الأساسية League Championship Algorithm (LCA)، ولكن تم تعزيزها لتدعم تعدد الأهداف (Multi-Objective Optimization) من خلال منهجية متقدمة لتقسيم الفرق وتحقيق التوازن بين عدة أهداف متضاربة. يهدف النموذج إلى حل مشكلتين أساسيتين في سلسلة الإمداد بشكل متزامن وهما جدولة الموردين وجدولة المركبات. ‎[6]

* + 1. جدولة الموردين (Supplier Scheduling):

تتمحور هذه العملية بكيفية جدولة إنتاج المواد الخام أو المكونات المطلوبة في المصانع التابعة للموردين. النموذج يقترح طرقًا لتحديد أي مورد سيقوم بإنتاج أي طلب، وبأي ترتيب زمني، مع مراعاة تقليل التأخير في الإنتاج.

ويكون الهدف تقليل التأخير في تسليم الطلبات (Total Tardiness).

* + 1. جدولة المركبات (Vehicle Scheduling):

يتم نقل الطلبات – بعد تجهيزها – إلى مراكز التصنيع عبر مركبات نقل. هنا تبرز مشكلة تحديد مسار كل مركبة، وترتيب عمليات الشحن لتقليل المسافات المقطوعة وتقليل وقت النقل، مما يؤدي إلى خفض التكاليف اللوجستية.

ويكون الهدف هنا تقليل المسافات المقطوعة بواسطة المركبات (Total Travel Distance).

* + 1. الخوارزمية المستخدمة Multiple League Championship Algorithm (MLCA):

صمّم الباحثون نسخة مطوّرة من خوارزمية LCA تسمى MLCA، المستوحاة من بطولات الدوري الرياضية. الفكرة الأساسية تعتمد على تنظيم الحلول في شكل "فرق" تتنافس في بطولات محلية، ثم تتأهل الفرق الأفضل إلى مراحل أعلى (بطولات قارية) حتى الوصول إلى الحل الأمثل.

الية العمل:

* تقسيم الفرق إلى دوريات (Leagues):

في البداية، يتم إنشاء مجموعة عشوائية من الحلول (Teams)، حيث يمثل كل حل ترتيباً محتملاً لجدولة الطلبات والمركبات.

تُوزع هذه الفرق على دوريات محلية (National Leagues).

كل دوري يمثل مساحة محددة من الحلول في فضاء البحث.

* المباريات داخل الدوري (Match Operator):

داخل كل دوري، تتنافس الفرق مع بعضها البعض، بحيث يتم تبادل الخبرات والاستراتيجيات لتحسين الأداء.

في كل مباراة، يتم تعديل بعض عناصر الحلول بناءً على أداء الفرق المنافسة.

إذا وجد تحسين في دالة الهدف، يتم الاحتفاظ بالتغيير؛ وإذا لم يكن هناك تحسن، يعود الحل لوضعه الأصلي.

* التقييم متعدد الأهداف (Multi-objective Evaluation):

بدلاً من تحسين دالة هدف واحدة، تعتمد MO-LCA على تقنيات لتقييم الحلول بناءً على Pareto Dominance، حيث يتم تصنيف الحلول إلى:

حلول Pareto Optimal، وهي الحلول التي لا يمكن تحسين أحد أهدافها دون التأثير سلباً على الأهداف الأخرى.

حلول غير مهيمنة (Non-Dominated Solutions) التي تتمتع بتوازن جيد بين الأهداف.

* التأهل إلى البطولات القارية (Continental Championships):

بعد انتهاء المباريات في الدوريات المحلية، يتم اختيار أفضل الحلول (الفرق التي تحقق أقل تأخير وأقصر مسافة) للتنافس في بطولات قارية.

في هذه المرحلة، يتم تحسين الحلول على نطاق أوسع مع فرق أخرى من دوريات مختلفة.

* مباريات التصفيات (Playoffs):

في هذه المرحلة النهائية، تتنافس الحلول القوية لتحديد "البطل"، أي أفضل حل متعدد الأهداف يحقق أقل تأخير وأقل مسافة مقطوعة في نفس الوقت.

* التكرار (Iterations):

تستمر الخوارزمية في التكرار حتى الوصول إلى شرط التوقف، مثل عدم تحسن الحلول خلال عدد معين من الأجيال، أو الوصول إلى حد أقصى لعدد التكرارات.

* + 1. تقنيات التقييم متعددة الأهداف:

لضمان تحقيق تحسينات في كلا الهدفين، استخدمت الخوارزمية طريقة **Weighted Sum Approach**، بحيث يتم دمج دالتي الهدف في دالة واحدة وفقاً للمعادلة (1):

حيث:

TAR: التأخير في التسليم.

DIS: المساقة المقطوعة.

W: أوزان تعكس أهمية كل هدف.

* + 1. مبدأ Pareto Optimality:

في كل جيل، يتم البحث عن مجموعة من الحلول التي تكون Pareto Optimal، مما يعني أنها تمثل توازنًا مثالياً بين الأهداف دون تفوق هدف على آخر بشكل مطلق. وهذا يتيح للمستخدم اختيار الحل الأنسب بناءً على المعايير المطلوبة.

* + 1. النتائج:

تختلف النتائج عند مقارنة الخوارزميات باختلاف مسائل الاختبار، فقد تظهر خوارزمية نتائج ممتازة عندما يكون عدد الطلبات قليل ونتائج سيئة عندما يكون عددها كبير، لهذا السبب أنشا الباحثون عدداً كبيراً من مسائل الاختبار، فكان عددها 300 مسالة موزعة على 10 أنواع من المسائل تختلف فيما بينها بقيم (عدد الطلبات، عدد الموردين، عدد المركبات وسعتها، الخ...).

أظهرت النتائج أن خوارزمية MLCA تتفوق على الخوارزميات التي تم المقارنة معها (dynamic genetic algorithm, LCA, golden ball, Ullrich genetic algorithm) لكنها تتطلب وقت حساب أكثر لإعطاء النتائج.

* 1. أعمال سابقة في جدولة المهام في بيئة سحابية

تعتبر مسألة جدولة المهام في بيئة سحابية من المسائل الصعبة (NP-Complete)، ولحل هذه المشكلة طورت العديد من الطرق التقديرية (Heuristic) والاستراتيجيات فوق التقديرية (Meta-Heuristic).

أهم الخوارزميات والتقنيات التي ظهرت لجدولة المهام:

أوصى كل من R. Kaur و S. Kinger باستخدام الخوارزمية الجينية (GA) لتحسين جدولة المهام. حيث تعتمد هذه الخوارزمية على قيم المتوسط والمتوسط العام (Mean and Grand Mean) في استخدام دالة ملائمة فريدة (Fitness Function). ‎[7]

اقترح Kokilavani وآخرون تقنية MinMin (LBMM) load balancing، والتي تهدف إلى تحسين استخدام الموارد مع تقليل "Makespan". تتكون هذه التقنية من مرحلتين:

في المرحلة الأولى، يتم تشغيل خوارزمية Min-Min التقليدية.

في المرحلة الثانية، يُعاد جدولة المهام لتحقيق استهلاك أكثر كفاءة للموارد. ‎[8]

طرق الجدولة فوق التقديرية (Meta-Heuristic) وتشمل كلًا من:

* محاكاة التلدين (Simulated Annealing - SA)
* الخوارزمية الجينية (Genetic Algorithm - GA)
* أمثلة سرب الجسيمات (Particle Swarm Optimization - PSO)
* أمثلة مستعمرة النمل (Ant Colony Optimization - ACO)

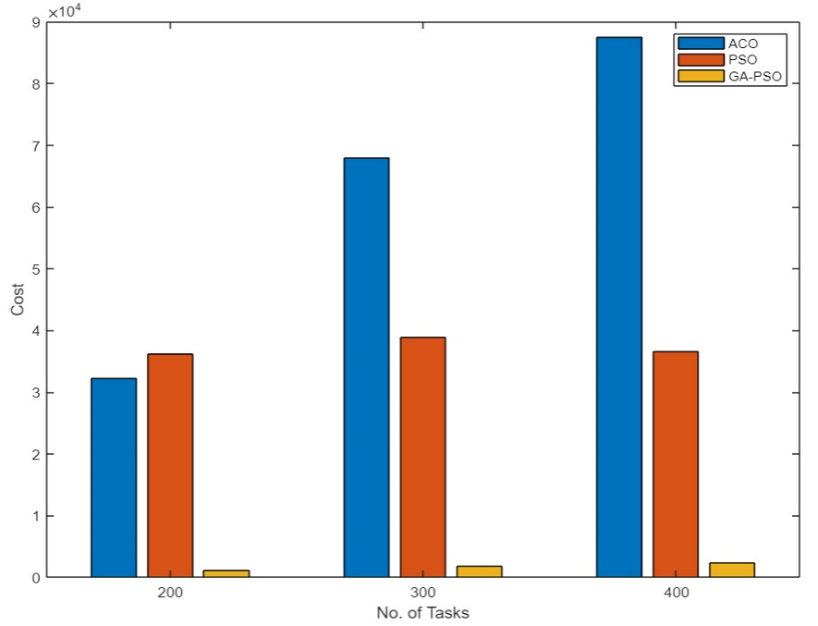
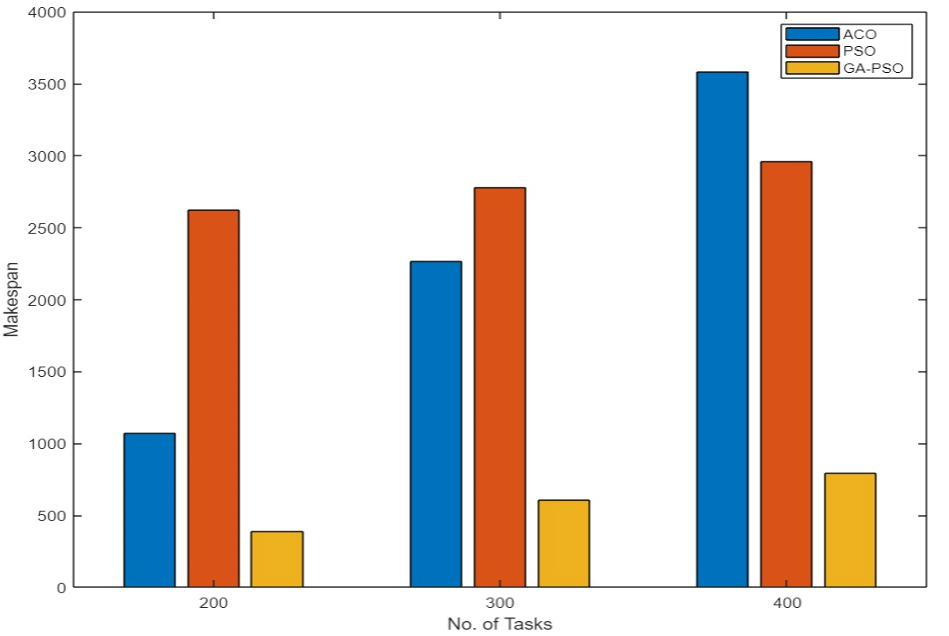
تستخدم خوارزمية **ACO** مجموعة من المبادئ المستوحاة من الطبيعة لحل مشكلات الأمثلة المعقدة. استخدم **Tawfeek وآخرون** مجموعة معايير **ACO** لجدولة المهام في بيئة الحوسبة السحابية، وتم تقييم فعالية الخوارزمية من خلال مقارنتها مع خوارزميات **Round-Robin** **وFCFS**. ‎[9]

قدم **S.Singh** عدة احتمالات لجدولة المهام في بيئة الحوسبة السحابية، حيث قدم فهماً شاملاً للخوارزمية الجينية **(GA)**. وقام بتطوير منهجية لمعالجة مشكلات جدولة العمل باستخدام طريقة **Max-Min** لبناء التجمع الأولي (Initial Population) وتحقيق أفضل النتائج فيما يتعلق بمقياس **Makespan** عبر تعديل الخوارزمية الجينية. كما وصف المؤلفون نموذجًا يعتمد على **Cuckoo Search** لتحسين أداء المهام في هذا العمل. ‎[10]

خوارزمية هجينة GA-PSO

تجمع الخوارزمية الهجينة **GA-PSO (Genetic Algorithm - Particle Swarm Optimization)** بين قدرة البحث الشامل في الخوارزمية الجينية (**GA**) مع خصائص التقارب السريع في خوارزمية تحسين سرب الجسيمات (**PSO**). في هذا النموذج الهجين، تكون مسؤولية **GA** بشكل أساسي هي الحفاظ على التنوع في فضاء البحث من خلال عمليات التهجين (Crossover) والطفرات (Mutation)، بينما تقوم **PSO** بتسريع البحث عن الحلول المثلى عبر تعديل سرعات الجسيمات بناءً على أفضل المواقع الشخصية والعالمية.

في سياق جدولة المهام في بيئات الحوسبة السحابية، تم استخدام **GA-PSO** لتعزيز كفاءة تخصيص الموارد وتقليل مدة التنفيذ الكلية (**Makespan**). من خلال الاستفادة من قدرة **GA** على الاستكشاف، وقوة **PSO** في الاستغلال الأمثل، يحقق هذا النهج الهجين توزيعًا متوازنًا للأحمال عبر الأجهزة الافتراضية (**Virtual Machines**). أظهرت الدراسات أن **GA-PSO** يتفوق على الطرق التقليدية المستقلة مثل **GA** و **PSO** من خلال تحقيق أوقات إتمام مهام أقل وتقليل التكاليف الإجمالية للتنفيذ. ‎[11]

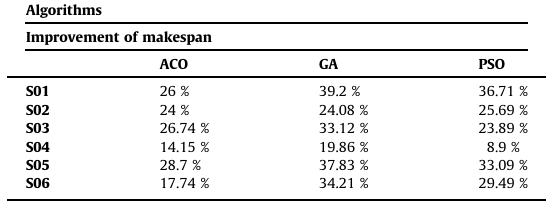
مقارنة بين أداء **GA-PSO** و**PSO** و**ACO:**

الشكل 4 مقارنة حسب مدة التنفيذ

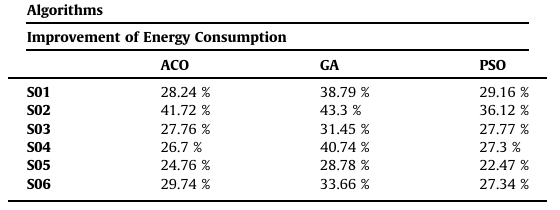
الشكل 5 مقارنة حسب الكلفة

وقد اقترح **Mangalampalli** وآخرون (2023) خوارزمية جدولة متعددة الأهداف واعية بالثقة **(Trust-aware Multi-objective Scheduling)** في بيئة الحوسبة السحابية. تقوم هذه الخوارزمية بأخذ ثقة الآلات الافتراضية (VMs) وأولوية المهام كعوامل أساسية ضمن عملية التخصيص، مما يُمكّنها من تحسين جودة الخدمة (QoS) وتقليل احتمالية تعيين المهام إلى موارد غير موثوقة. تعتمد الخوارزمية على تخصيص ديناميكي للمهام بهدف موازنة الحمل وزيادة كفاءة استخدام الموارد، مع تحقيق تحسينات ملحوظة في الأداء من حيث Makespan واستهلاك الطاقة. وقد أظهرت نتائج التجارب أن المنهجية المقترحة تتفوق على الخوارزميات التقليدية من خلال تحقيق توزيع أكثر استقرارًا للمهام وتحقيق مستويات أعلى من رضا المستخدم النهائي. ‎[12]

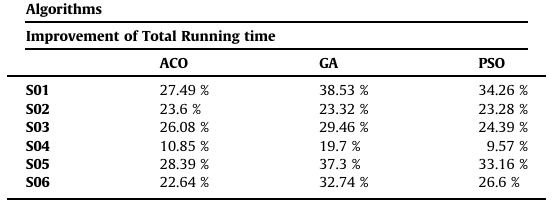
وبمقارنة أدائها مع خوارزميات أخرى أظهرت تحسينات ملحوظة على عدة جوانب أهمها:



الجدول 2 تحسينات من ناحية وقت التنفيذ الكلي

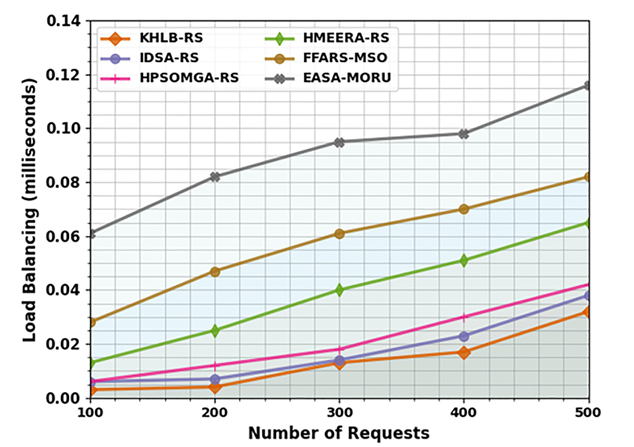


الجدول 3 تحسينات من نحاية استهلاك الطاقة

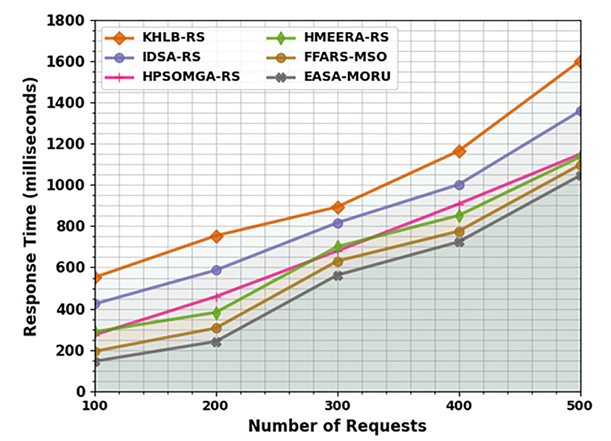


الجدول 4 حسينات من ناحية وقت التشغيل

كذلك، قدم Karim وآخرون (2024) خوارزمية تطورية حديثة تُعرف باسم EASA-MORU (Evolutionary Approach for Scheduling and Allocation – Multi-Objective Resource Utilization)، والتي تعتمد على خوارزمية تحسين مستوحاة من سلوك خنفساء الروث Dung Beetle Optimization (DBO). تهدف هذه الخوارزمية إلى تحسين استخدام الموارد وتقليل مدة التنفيذ الكلية، مع تحقيق توازن ديناميكي للأحمال في بيئة الحوسبة السحابية. وقد بينت نتائج التجارب تفوق EASA-MORU على الطرق التقليدية مثل KH-LBRS وFFA-RSMSO من حيث زمن الاستجابة، كفاءة استخدام الموارد، واستهلاك الطاقة، كما تظهر المخططات: ‎[13]



الشكل 6 مقارنة توزيع الحمل



الشكل 7 مقارنة زمن الاستجابة

الفصل الرابع

1. الدراسة التحليلية

نستعرض في هذا الفصل تحليل شامل لخوارزمية الحل المقترحة لجدولة المهام في بيئة سحابية.

* 1. دراسة المصطلحات ذات الصلة ببطولات الدوري الرياضي

نشرح في هذا القسم أهم المصطلحات التي تستخدم عادة في بطولات الدوري الرياضي.

الدوري: هو مجموعة من الفرق الرياضية التي تتنافس ضد بعضها في رياضة معينة، ويستخدم المصطلح بشكل عام للإشارة الى المنافسات التي تشمل الرياضات الجامعية وليس الفردية. يمكن أن يكون التنافس على بطولة الدوري بعدة طرق. يمكن أن يلعب كل فريق مع كل فريق آخر عدداً معيناً من المرات في بطولة من دور واحد. وفي مثل هذا الإعداد، يصبح الفريق صاحب أفضل سجل هو البطل، استناداً إما إلى نظام صارم يقوم على الفوز والخسارة والتعادل أو على نظام النقاط حيث يتم منح عدد معين من النقاط للفوز أو الخسارة أو التعادل.

الفريق: هو مجموعة من اللاعبين، يكون لكل لاعب دور محدد بناءً على تشكيل الفريق، هذا التشكيل هو هيكل محدد يحدد توزيع اللاعبين بناءً على مراكزهم ويعبر عن أسلوب لعب الفريق (هجومي، دفاعي)، ويتبع كل فريق أفضل تشكيلة والتي غالباً ما ترتبط بنوعية اللاعبين المتاحين للمدرب.

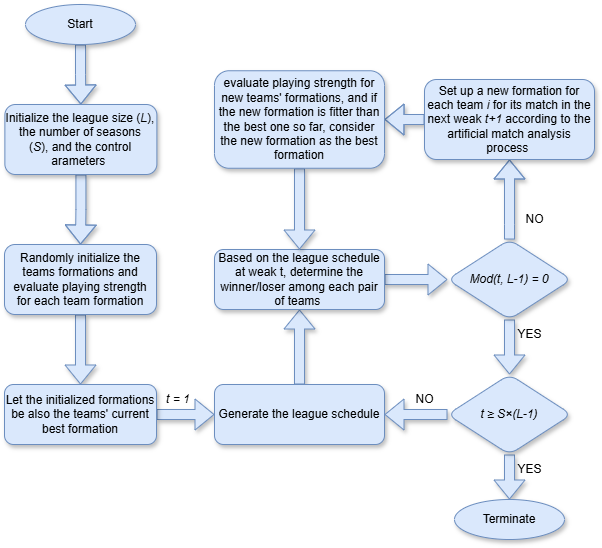
تحليل المباراة: يقوم المدرب بتحليل المباريات التي لعبها الفريق، الهدف منها هو مراقبة أداء الفريق وذلك لتحديد نقاط القوة والضعف لدى الفريق. أو يثوم المدرب بتحليل أداء الفريق الخصم في مبارياته السابقة وذلك لتحديد نقاط قوة الخصم (التهديدات) لمواجهتها ونقاط ضعف الخصم (الفرص) لاستغلالها.

ويُعرف هذا النوع من التحليل عادةً باسم تحليل نقاط القوة والضعف والفرص والتهديدات (SWOT)، يدرس هذا التحليل نوعين من العوامل:

* العوامل الداخلية وهي نقاط القوة والضعف
* العوامل الخراجية وي الفرص والتهديدات

وبناءً على هذا التحليل يحدد المدرب تشكيل الفريق في مباراته القادمة.

* 1. خوارزمية MO-LCA
     1. مخطط دفق الخوارزمية



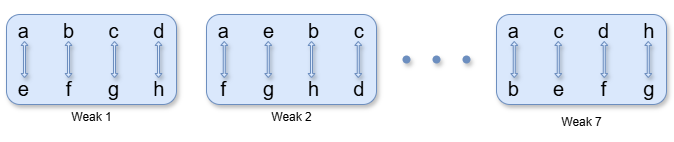
الشكل 8 مخطط تدفق خوارزمية MO-LCA

الاختلاف بين LCA وMO-LCA هو أن دالة الملائمة في LCA ينتمي للفضاء بينما في MO-LCA ينتمي للفضاء حيث هي عدد الأهداف التي نريد تحسينها.

* + 1. جدولة مباريات الدوري league schedule

عند إنشاء جدول مباريات الدوري يجب إنشائه بحيث يتنافس كل فريق ضد بقية الفرق الأخرى بناءً على جدول زمني محدد، يسمى عادةً الموسم. وفي خوارزمية LCA يتم استخدام جدول بحيث يلعب كل فريق مع كل الفرق الأخرى مرة واحدة في كل موسم. وفي دوري رياضي يتألف من فريق، ستجرى مباراة، لأنه في كل أسبوع من أصل أسبوع، ستجرى مباراة على التوازي (إذا كانت فردية فستكون هناك جولة مع مباراة، ولن يكون لفريق واحد مباراة في تلك الجولة).

مثال على league schedule لنعتبر انه لدينا 8 فرق (a…h) نثبت فريق في مكانه والبقية ندورهم على 7 أسابيع بهذا الشكل كل فريق يلعب مع جميع الفرق الاخرى



الشكل 9 مخطط جدولة مباريات الدوري

* + 1. دالة الملائمة Fitness function

يمكن تمثيل دالة الملائمة في خوارزمية MO-LCA كما في المعادلة (2):

أي أن قوة الفريق لا تقتصر على قيمة واحدة وإنما أكثر من قيمة (مثلا قوة هجومية، قوة دفاعية إلخ.)

الطريقة التقليدية للتعامل مع تعدد الأهداف هي بإنزال رتبة المسالة بطريقة ما إلى هدف واحد، مثلا مجموع موزن كما هو موضح في المعادلة رقم (3):

* + 1. تحديد الفائز في المباراة

في بطولة الدوري، تتنافس الفرق فيما بينها على كل أسبوعي ويتم تحديد نتيجة المباراة من حيث الفوز أو الخسارة أو التعادل لكل فريق. وبهذه الطريقة، على سبيل المثال في كرة القدم، يتم احتساب 3 نقاط لكل فريق في حالة الفوز، و0 للخسارة و1 للتعادل. ومن البديهي أن فرصة فوز الفريق ذي قوة اللعب الأعلى أعلى من فرصة فوز خصمه، ولكن لا تنتفي فرص فوز الفريق الأضعف، فيمكن اعتبار وجود ارتباط خطي بين قوة اللعب ونتيجة المباراة.

وباستخدام معيار قوة لعب الفريق، يتم تحديد الفائز/الخاسر في LCA بطريقة عشوائية مع الشرط الذي ينص على أن فرصة فوز فريق ما تتناسب مع درجة لياقته (تذكر أنه في النسخة الأساسية من قاعدة LCA لا توجد نتيجة تعادل). تتناسب درجة لياقة الفريق مع قوة لعبه وتقاس عن طريق المسافة مع نقطة مرجعية مثالية.

فبفرض فريقين و يمكن التعبير عن احتمال فوز كل من الفريقين في الأسبوع على الشكل التالي:

حيث و هو تشكيل الفريقين و في الأسبوع ، و هو قوة لعب الفريقين و في الأسبوع ، هو نقطة مرجعية مثلى تسعى الفرق للوصول لها، هو احتمال فوز الفريق على الفريق في الأسبوع ،و هو احتمال فوز الفريق على الفريق في الأسبوع . وفي حالة عدم السماح بوجود تعادل فإن

فيمكن كتابة على الشكل:

ولتحديد الفائز في المباراة، يتم توليد رقم عشوائي على المجال ، فاذا كان الرقم الناتج أصغر أو يساوي فإن الفريق يفوز على الفريق . وإذا كان أكبر فإن الفريق يفوز على الفريق في الأسبوع .

* + 1. تحليل المباريات

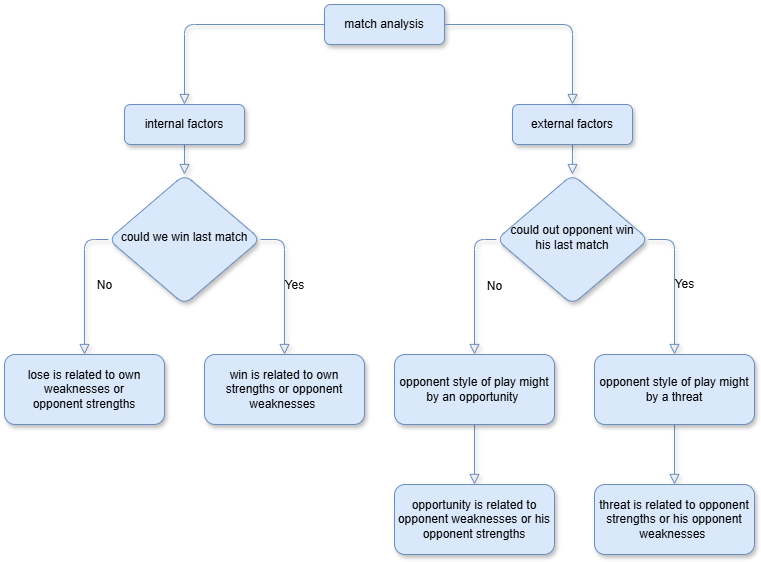
يقوم المدربون بتحليل المباريات في الأسبوع السابق لتشكيل الفرق في هذا الأسبوع حسب مصفوفة SWOT وذلك في محاولة منهم لزيادة قوة اللعب للفريق، حيث يحللون العوامل الداخلية (internal factors) التي تركز على نتائج مباراة الفريق في الأسبوع السابق والعوامل الخارجية (external factors) التي تركز على نتائج مباراة الفريق الخصم في الأسبوع السابق.

لتحليل مباريات للفريق في الأسبوع لوضع تشكيل جديد في الأسبوع ، إذا كان قد فاز (خسر) الفريق المباراة المصطنعة على (أمام) الفريق في الأسبوع ، فإننا نفترض أن النجاح (الخسارة) كان نتيجة مباشرة عن نقاط قوة (نقاط ضعف) الفريق ،أو نتيجة مباشرة عن نقاط ضعف (نقاط قوة) الفريق . واستناداً إلى جدول الدوري في الأسبوع، افترض أن المباراة التالية للفريق هي مع الفريق ، إذا كان الفريق قد فاز (خسر) المباراة على (أمام) الفريق في الأسبوع ، فإن هذا النجاح (الخسارة) وتشكيل الفريق الذي يقف وراءه قد يكون تهديداً مباشراً (فرصة) للفريق . من الواضح أن مثل هذا النجاح (الخسارة) ناتج عن بعض نقاط القوة (نقاط الضعف). إن التركيز على نقاط القوة (نقاط الضعف) في الفريق ، يعطينا طريقة بديهية لتجنب التهديدات المحتملة (للحصول على فوائد من الفرص المحتملة). أو يمكننا التركيز على نقاط الضعف (نقاط القوة) في الفريق بدلاً من ذلك. يوضح الشكل 9 ألية تحليل المباريات.

يمكن تلخيص الإجراءات الممكنة للفريق المستندة على تحليل المباريات حسب مصفوفة SWOT الموضحة في الجدول 5.

* تظهر إجراءات S/T نقاط القوة في ضوء التهديدات الرئيسية من المنافسين. يجب أن يستخدم الفريق نقاط قوته لتجنب التهديدات.
* تظهر إجراءات S/O نقاط القوة والفرص. يجب أن يحاول الفريق استخدام نقاط قوته لاستغلال الفرص.
* تظهر إجراءات W/O نقاط الضعف في مواجهة التهديدات القائمة. يجب أن يحاول الفريق التقليل من نقاط ضعفه وتجنب التهديدات. هذه الاستراتيجيات دفاعية بشكل عام.
* تظهر إجراءات W/T نقاط الضعف إلى جانب الفرص الرئيسية. يجب أن يحاول الفريق التغلب على نقاط ضعفه من خلال استغلال الفرص.

يتم إجراء هذا التحليل على جميع الفرق.الشكل 7



الشكل 10 تحليل المباريات

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| had lost  had lost  Focusing on … | had lost  had won  Focusing on … | had won  had lost  Focusing on … | had won  had won  Focusing on … |  |
| - | - | Own strengths  (or weaknesses of ) | Own strengths  (or weaknesses of ) | S |
| Own weaknesses  (or strengths of ) | Own weaknesses  (or strengths of ) | - | - | W |
| weaknesses of  (or strengths of ) | - | weaknesses of  (or strengths of ) | - | O |
| - | strengths of  (or weaknesses of ) | - | strengths of  (or weaknesses of ) | T |

الجدول 5 مصفوفة SWOT

دعونا نقدم المؤشرات التالية:

= مؤشر الفريق الذي سيلعب مع الفريق في الأسبوع استناداً الى جدول الدوري.

= مؤشر الفريق الذي لعب مع الفريق في الأسبوع استناداً الى جدول الدوري.

= مؤشر الفريق الذي لعب مع الفريق في الأسبوع استناداً الى جدول الدوري.

وذلك لجميع الفرق .

لنفرض أن و و هي تشكيلات الفرق و و في الأسبوع ، على التوالي. ونعني مثلا بـ بالفجوة بين أسلوب اللعب للفريقين و والتي سيتم استشعارها من خلال "التركيز على نقاط قوة الفريق ". في هذه الحالة، يكون الفريق قد فاز بالمباراة على الفريق ، وللتغلب على ، فمن المنطقي أن يبتكر الفريق أسلوب لعب مشابه تقريبًا للأسلوب الذي اعتمده الفريق في الأسبوع (على سبيل المثال، اللعب بأسلوب الهجوم المرتد أو الدفاع عالي الضغط). وبطريقة مماثلة يمكننا تفسير عند "التركيز على نقاط ضعف الفريق ". هنا، قد يكون من المنطقي تجنب أسلوب لعب مشابه إلى حد ما لأسلوب لعب الفريق (على سبيل المثال، تجنب اللعب بالهجوم المضاد أو الدفاع عالي الضغط). يمكننا تفسير أو  بطريقة مماثلة.

وبالنظر إلى حقيقة أن الفرق عادةً ما تلعب بناءً على أفضل تشكيلاتها الحالية (وجدت أنها مناسبة على مدار الوقت) مع إعداد التغييرات المطلوبة التي أوصى بها تحليل المباراة، فإن التشكيل الجديد للفريق في الأسبوع يمكن إعداده بإحدى المعادلات التالية:

إذا فاز وفاز ، يتم إنشاء التشكيل الجديد بناءً على تكييف استراتيجية S/T:

إذا فاز وخسر ، يتم إنشاء التشكيل الجديد بناءً على تكييف استراتيجية S/O:

إذا خسر وفاز ، يتم إنشاء التشكيل الجديد بناءً على تكييف استراتيجية W/T:

إذا خسر وخسر ، يتم إنشاء التشكيل الجديد بناءً على تكييف استراتيجية W/O:

حيث هو عدد أعضاء الفريق، و هما معاملان يستخدمان لتحديد مساهمة مكوني "التراجع" أو "الاقتراب" على التوالي، و هما رقمان عشوائيان على المجال ، هو متغير ثنائي يشير إلى ما إذا كان العنصر في التشكيل الجديد يجب أن يختلف عن نظيره في التشكيل الأفضل الحالي أم لا (فقط يسمح بالاختلاف).

لاحظ أن علامة الفرق الموجودة بين قوسين تؤدي إلى التسارع نحو الفائز أو التراجع عن الخاسر.

لنعرف هو متجه التغيير الثنائي، وليكن عدد الواحدات فيها . ليس من المعتاد أن يقوم المدربون بإجراء تغييرات على كافة جوانب الفريق أو العديد منها، يوصى بعدد قليل من التغييرات. لحساب عدد التغييرات التي نريدها، نستخدم توزيع احتمالي هندسي مبتور (truncated geometric distribution) للتحكم في عدد التغييرات التي سنجريها في مع التركيز على المعدل الأصغر للتغييرات. نعبر عنها بالمعادلة:

حيث، هو رقم عشوائي على المجال ، هو معامل تحكم بحيث ، هو الحد الأدنى من التغييرات. إذا كانت قيمة سالبة فيكون الوضع معكوساً وكلما كانت قيمتها سالبة أكثر كلما زاد التركيز على معدل التغييرات الأكبر. يتم تحديد أثناء تحليل المباراة (لاحظ أن الحد الأدنى لقيمة هو الصفر).

بعد حساب قيمة ، يتم اختيار عنصر من عشوائياً وتتغير قيمتها وفقاً لأحد معادلات الاستراتيجيات السابقة.

* + 1. Pseudo Code

Begin

Initialize the league size and the number of seasons

Initialize teams by generating random schedule with population size of

Evaluate the objective function and assign it as playing strengths of each team

Initialize the best formation to initial formation

Generate a league \*timetable

For to

Based on the league \*timetable determine winner/loser for each pair of teams

For to

Set up new formation for team based on SOWT analysis

Update the best formation (if needed)

End for

If

Generate a league \*timetable

End if

End for

(Note: \* indicates the word timetable is used instead of the word schedule in MO-LCA to

avoid confusion.)

* 1. دوال الهدف objective functions

تعرف دوال الهدف العلاقات الرياضية لحساب التكلفة (cost) ووقت التنفيذ (makespan) من أجل حل X.

هنالك عدة عوامل تؤثر في حساب دوال الهدف، ففي حالة حساب الكلفة اللازمة لتنفيذ المهام على الآلات الافتراضية، يجب الأخذ بعين الاعتبار استهلاك الطاقة في حالتي النشاط والخمول وتكلفة حجز الموارد (تختلف من آلة لأخرى حسب سرعة أدائها) وقد يحاسب مزودي الخدمة أسعار مختلفة للآلات الافتراضية تبعا لل Datacenter الذي يستضيفها.

أما بالنسبة لوقت التنفيذ فهو ‬

إن تقليل هذا الزمن قدر الإمكان هو هدف ‬

تقنية‬

* + 1. وقت التنفيذ (makespan)

يحسب وقت التنفيذ من خلال تحديد الزمن النهائي لأخر مهمة مكتملة، كما في المعادلة (12):

حيث هو وقت انتهاء المهمة .

و هو وقت انتهاء تنفيذ جميع مهام الآلة الافتراضية .

ففي حالة استخدام تقنية مشاركة المساحة، يتم حساب وقت انتهاء المهمة  كما في المعادلة (13):

حيث هو وقت تنفيذ المهمة على الآلة الافتراضية .

و هو وقت انتظار المهمة حتى تتوافر الموارد اللازمة لتشغيلها على الآلة الافتراضية .

أما في حالة استخدام تقنية مشاركة الزمن، يتم حساب وقت التنفيذ بشكل تقريبي، كما في المعادلة (14):

حيث يتعلق بعدد المهام التي يتم تبديلها على التناوب على الألة الافتراضية وعدد وحدات المعالجة فيها.

و هو Context Switch Overhead الزمن الإضافي الناتج عن عملية تبديل المهام.

* + 1. حساب الكلفة (Cost)

نعرف أولا أن كل افتراضية لها استهلاك طاقة في حالتيها النشطة والخاملة:

يحسب استهلاك الطاقة لكل آلة افتراضية، باستخدام المعادلة (16):

ثم يحسب مكون الطاقة الفعال (Active Energy Component) وفق المعادلة (17):

حيث هو معامل استخدام الموارد.

وأخيرا يتم حساب الكلفة باستخدام المعادلة (18):

* + 1. تنفيذ دالة الهدف في MO-LCA

نعتمد طريقة **Weighted Sum Approach**:

حيث يحدد المعامل إن كان التركيز على أمثلة وثت التنفيذ أو امثلة الكلفة.

مع ملاحظة أنه يجب الانتباه إلى أن الكلفة ووقت التنفيذ ليس لهم نفس المقياس، حيث سيتم حل هذه المشكلة في فقرة لاحقة بأخذ نسبة مئوية من حل مرجعي.

الفصل الخامس

1. الأدوات والبيئات المستخدمة

نستعرض في هذا الفصل البيئات والأدوات المستخدمة ومقارنتها مع أدوات أخرى.

* 1. بيئة المحاكاة السحابية CloudSim Plus

تُعد أدوات محاكاة البيئات السحابية عنصراً أساسياً في البحث والتطوير، حيث تتيح اختبار النماذج وتقييم الأداء دون الحاجة إلى بنية تحتية مادية. ومن بين هذه الأدوات، يبرز CloudSim Plus كإطار عمل متطور يوفر ميزات متقدمة مقارنةً بالأدوات التقليدية مثل CloudSim وGreenCloud. وفيما يلي مقارنة تحليلية تُظهر مميزات وعيوب كل أداة، حيث يتفوق CloudSim Plus في دعمه للتقنيات الحديثة وسهولة استخدامه.

|  |  |  |
| --- | --- | --- |
| الأداة | المميزات | العيوب |
| CloudSim Plus | واجهة برمجة مبسطة وسهلة التوسع. يدعم **الحاويات، الطاقة، والحوسبة الطرفية**.  نماذج دقيقة لاستهلاك الطاقة والكلفة. تحديثات مستمرة ومجتمع نشط. أمثلة توضيحية غنية ووثائق شاملة. | يحتاج إلى خبرة أولية في Java. غير مناسب لمحاكاة الشبكات المعقدة |
| CloudSim | أساس قوي لمحاكاة السحابة. متوافق مع أبحاث قديمة. | واجهة معقدة تتطلب أكوادًا طويلة. لا يدعم تقنيات حديثة (مثل الحاويات). محدودية في نمذجة الطاقة. |
| GreenCloud | متخصّص في **كفاءة الطاقة**. يحاكي مراكز البيانات بدقة. | قديم وغير محدّث. لا يدعم محاكاة الكلفة.  وثائق غير كافية. |
| iCanCloud | يركز على **التكلفة والأداء**. يدعم محاكاة التخزين. | محدودية في التخصيص. توقف التطوير منذ 2016. |

الجدول 6 مقارنة بين أدوات محاكاة السحابة

يعتبر **CloudSim Plus** الخيار الأمثل بسبب دعمه لنماذج استهلاك الطاقة والكلفة، بالإضافة لتوفر الأمثلة التوضيحية المفصلة والوثائق الشاملة.

* 1. اللغات البرمجية المستخدمة
     1. Java

لغة برمجة كائنية التوجه عالية المستوى متعددة الاستخدامات، تتميز بقوة الأداء وإدارة الذاكرة التلقائية. تعتمد على مبدأ "اكتب مرة واحدة، شغل في أي مكان" (WORA) مما يجعلها مناسبة للتطبيقات Cross-platform. في هذا البحث، تم استخدام جافا لتنفيذ جزء المحاكاة السحابية نظراً لكونها اللغة الأساسية لأطر عمل Cloudsim-plus. ويستفيد من كفاءة جافا في إدارة العمليات المتزامنة والمعقدة. كما تتيح جافا إمكانية التوسع بإضافة الصفوف الخاصة حسب متطلبات المشروع.

* + 1. Python

لغة برمجة عالية المستوى وسهلة التعلم، تشتهر ببساطة تركيبها اللغوي وغنى مكتباتها. تدعم أنماطاً متعددة للبرمجة بما في ذلك الكائنية والوظيفية. تم في هذا البحث توظيف بايثون بشكل رئيسي لكتابة الخوارزميات المتخصصة في جدولة المهام وتحسين الأداء، مثل خوارزميات MO-LCA وACO، وكذلك لتحليل النتائج المستخلصة من المحاكاة. تستفيد البرمجة بـ Python من المكتبات العلمية المتقدمة مثل NumPy للتحليل العددي وإدارة البيانات.

* + 1. React

هي مكتبة JavaScript مفتوحة المصدر، تُستخدم لبناء واجهات المستخدم، خاصة لتطبيقات الويب. تم تطويرها بواسطة شركة Facebook، وتستخدم على نطاق واسع لإنشاء واجهات مستخدم ديناميكية وتفاعلية. تتبع React البنية القائمة على المكونات، مما يساعد المطورين على بناء مكونات واجهة المستخدم التي يمكن إعادة استخدامها وتكوينها معًا لإنشاء واجهات مستخدم معقدة.

* + 1. Node.js

هي بيئة تشغيل مفتوحة المصدر ومتعددة المنصات تستخدم بشكل أساسي لتطوير تطبيقات الخوادم، تم استخدامها لتطوير الواجهة الخلفية (Backend) الخاصة بتطبيق الويب، مما مكن من إنشاء خادم ويب قادر على استقبال الطلبات من المستخدم، والتعامل مع تشغيل الخوارزميات عبر استدعاءات النظام، وإدارة عمليات الإدخال والإخراج.

* + 1. Javascript

جافا سكريبت هي لغة برمجة عالية المستوى تُستخدم أساساً لتطوير تطبيقات الويب التفاعلية، حيث تعمل على جانب العميل (Client-side) ضمن متصفحات الإنترنت. تُعرف جافا سكريبت بقدرتها على التفاعل الديناميكي مع عناصر صفحة الويب، وتمكين تطوير واجهات مستخدم غنية وتجارب استخدام سلسة. في هذا البحث، تم توظيف جافا سكريبت في تطوير واجهة المستخدم من خلال مكتبة React.js، وفي تطوير الواجهة الخلفية (Backend) الخاصة بتطبيق الويب من خلال بيئة Node.js.

* + 1. Git

هو نظام تحكم في الإصدارات (version control system) مفتوح المصدر يستخدم لتتبع التغييرات في الرماز المصدري (source code) أثناء تطوير البرمجيات. حيث يسمح Git لمطورين متعددين بالتعاون في مشروع واحد من خلال إدارة التغييرات التي تم إجراؤها على الرماز المصدري، مما يجعل من السهل تنسيق العمل وتتبع التغيرات ضمن المشروع.

* + 1. GitHub

هي منصة تعمل ضمن الويب توفر أدوات للتحكم في الإصدارات والتعاون بين المطورين لتطوير البرامج، حيث أنها مبنية على نظام التحكم في الإصدارات Git. تسمح هذه المنصة بتخزين الكود البرمجي ومشاركته بين مجموعة من المطورين كما تقدم مجموعة من الأدوات للتعامل مع بيئات التطوير المستندة إلى السحابة (cloud) وإستضافة مواقع الويب الثابتة.

الفصل السادس

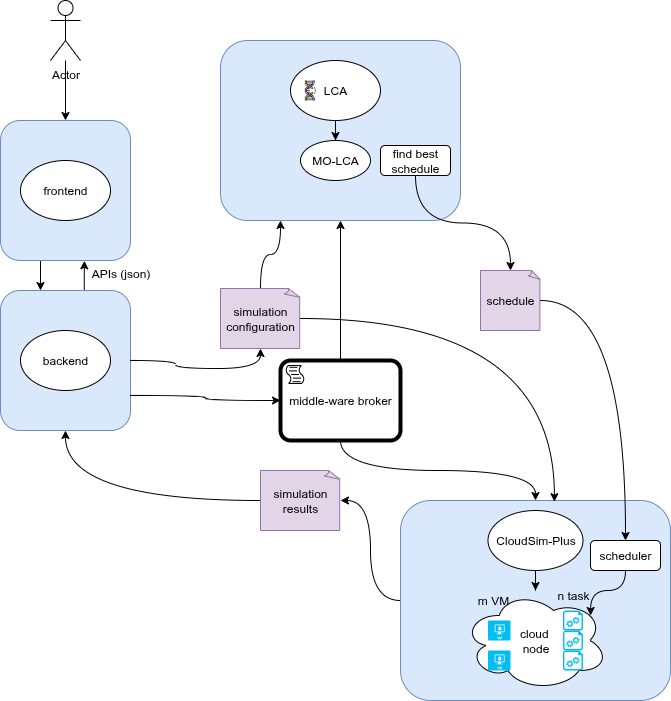
1. تصميم النظام

نستعرض في هذا الفصل تصميم النظام والمشاكل التي واجهتنا وكيفية تعاملنا معها.

* 1. هيكلة المشروع

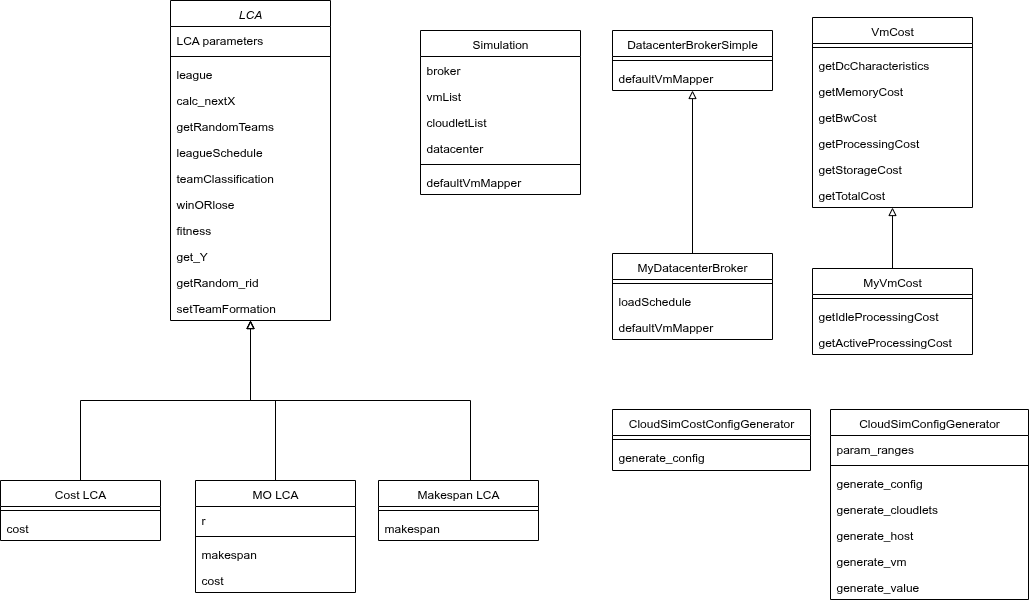
تم تصميم المشروع وفق بنية معيارية Modular Architecture with Orchestration تسهّل تطوير خوارزمية بطولة الدوري متعددة الأهداف (MO-LCA) ودمجها مع بيئة محاكاة سحابية مرنة. يتكون المشروع من المكونات الأساسية التالية:

* **وحدة الخوارزمية (Algorithm Module):** تحتوي على تنفيذ خوارزمية بطولة الدوري الأساسية بهدف واحد وبعدة أهداف، مع إمكانية تعديل توابع الهدف لتتناسب مع خصائص بيئة السحابة ومتطلبات الجدولة. كما تحتوي هذه الوحدة على تنفيذ خوارزميات جدولة أخرى ليتم المقارنة معها.
* **بيئة المحاكاة (Simulation Environment):** تعتمد على مكتبة CloudSim-Plus لمحاكاة بيئة الحوسبة السحابية، بما في ذلك إدارة الموارد، توزيع المهام، وحساب مؤشرات الأداء مثل زمن التنفيذ الكلي والكلفة.
* **واجهة الويب (Web Interface):** تتيح للمستخدمين تهيئة بيئة المحاكاة ومعاملات الخوارزمية عبر واجهة رسومية سهلة الاستخدام، يمكن من خلالها اختيار الخوارزميات المراد تشغيلها للمقارنة بينها، بدء عملية الجدولة، واستعراض النتائج الناتجة بشكل مباشر. هذه الوحدة بُنيت باستخدام تقنيات ويب حديثة لتبسيط تجربة الاستخدام وتسهيل إجراء التجارب.
* **ملفات الإعدادات (JSON Files):** تُستخدم للتخزين المؤقت لمعاملات المحاكاة والخوارزمية مثل عدد المهام، عدد الأجهزة الافتراضية وسرعاتها، ونموذج حساب الكلفة، مما يسمح بفصل البيانات عن المنطق البرمجي وتسهيل التعديل لاحقًا، بالإضافة لتخزين نتائج الخوارزميات ونتائج المحاكاة.
* **واجهة التكامل (middle-ware broker):** أو ال Orchestrator تقوم بربط الوحدات المختلفة فيما بينها بحيث لا تعتمد وحدة على أخرى بشكل مباشر باستثناء ملفات الإعداد ومخرجات الخوارزميات المحفوظة بصيغة JSON التي ترتبط بشكل مباشر مع مدخلات بيئة المحاكاة.

نستعرض في الشكل 11 مخطط صندوق النظام:

الشكل 11 مخطط صندوق النظام

* 1. مخطط الصفوف Class diagram

نستعرض في الشكل 12 مخطط الصفوف:

الشكل 12 مخطط الصفوف

حيث تتكون LCA parameters من المعاملات التالية: PSI1, PSI2, q0, p\_c, L, S, n

الصفان CloudSimConfigGenerator وCloudSimCostConfigGenerator مسؤولان عن تهيئة بيئة المحاكاة.

الصف LCA والصفوف التي ترثه هي الصفوف المسؤولة عن جدولة المهام (تم إضافة صفوف لخوارزميات إضافية لم يتم إدراجها في مخطط الصفوف).

الصفوف Simulation, MyVmCost, MyDatacenterBroker هي المسئولة عن تشغيل المحاكاة بالاعتماد على بيئة CloudSim-Plus.

* 1. تنفيذ وحدة الخوارزمية

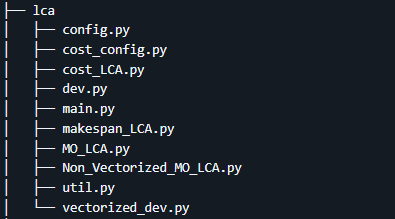
في تصميم وحدة الخوارزمية تم اعتماد بنية Plugin Architecture.

حيث يقوم ال Orchestrator باستدعاء main.py الذي يمثل entry point يقوم بتشغيل الخوارزمية المطلوبة بشكل ديناميكي.

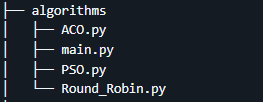
فوائده:

* Flexibility: السماح بإضافة سمات جديدة بدون التعديل على النظام.
* Scalability: السماح بإضافة خوارزميات جديدة و نسخ عن الخوارزميات الموجودة.
* Modularity and Maintainability

وتكون بنية الملفات في هذه الوحدة كالتالي:



بالإضافة إلى قسم يحتوي الخوارزميات التي يتم المقارنة معها:



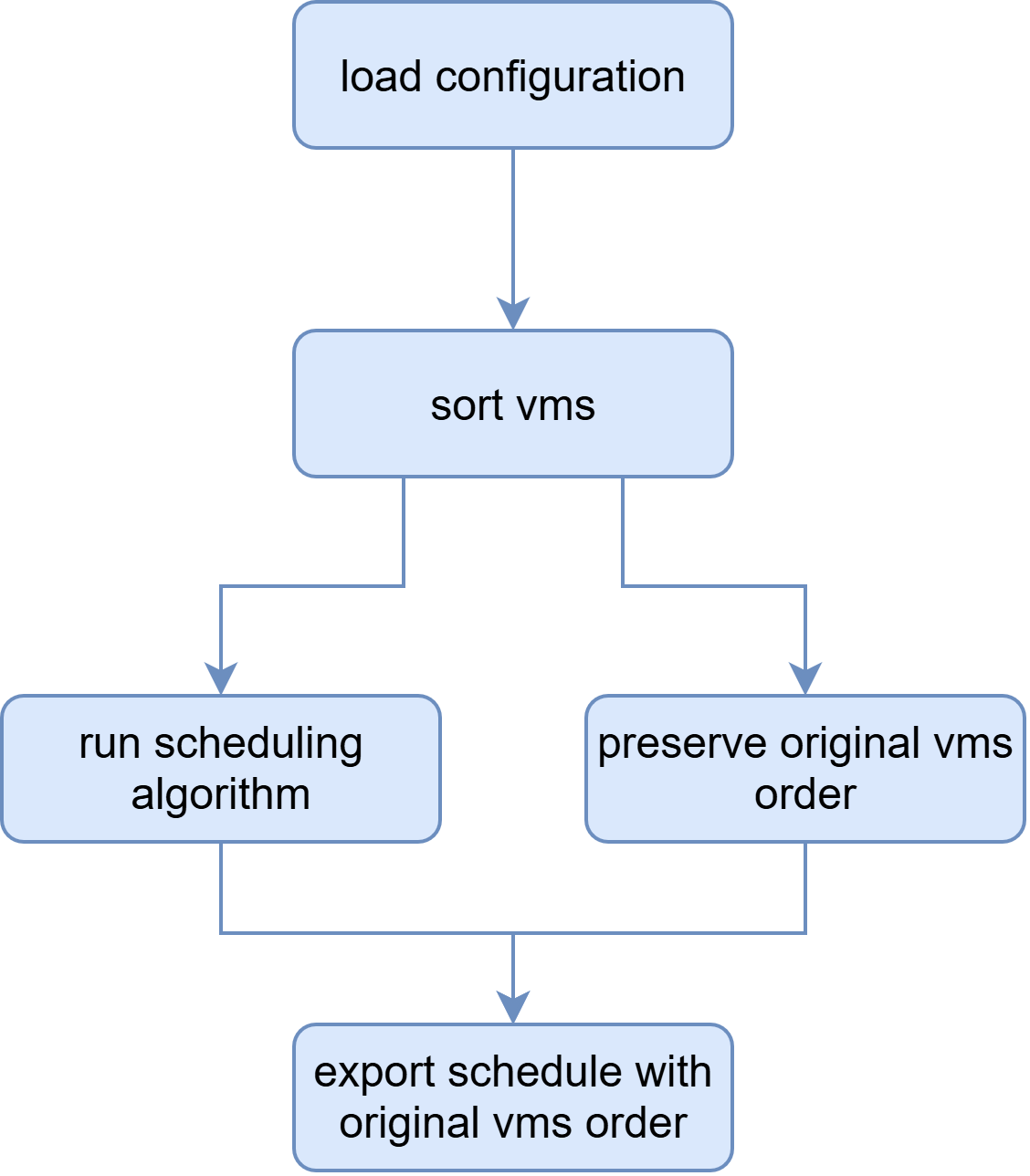
في البداية تم تطوير نسخة أولية من خوارزمية LCA مع تابع هدف عشوائي لاختبار بنية الكود والتحقق من صحة العمليات الأساسية. بعد ذلك، تم توريث هذه النسخة في صفوف فرعية وتعديل توابع الهدف بما يتناسب مع متطلبات المشروع.

في النسخة الأولية تم تنفيذ التوابع المساعدة في عل الخوارزمية مثل إنشاء حلول أولية عشوائية، حساب الفائز بين فريقين، القيام بتحليل SOWT، إنشاء جدول المباريات وغيرها من التوابع.

عند إنشاء جدول المباريات ولكيلا يتم إنشاء نفس جدول اللعب في كل موسم، تم إنشاء جدول عام يتم اعتماده في أول موسم، ثم يتم تغيير ترتيب الفرق عشوائيا واسنادهم الى جدول اللعب العام، بهذه الطريقة يتم في كل موسم يكون جدول اللعب عشوائي، مثال عن ذلك في حالة 4 فرق:



عند تنفيذ التابع المسؤول عن عملية تحليل SWOT، ظهرت تحديات تتعلق بمفهوم العمليات الجينية المستخدمة في المعادلات الرياضية فيها، وبالتحديد عملية طرح الكروموسومات التي تتطلب وجود معنى منطقي لهذه العملية. وللتغلب على ذلك، تم اعتماد أسلوب ترتيب الأجهزة الافتراضية وفق سرعاتها، بحيث يؤدي زيادة قيمة الجين في الكروموسوم إلى إسناد المهمة إلى آلة افتراضية ذات سرعة أعلى (أو العكس في حال نقصان القيمة). تم تنفيذ عملية ترتيب الآلات الافتراضية بشكل منفصل عن منطق الخوارزمية وذلك لإتاحة الفرصة لتغيير سياسة ترتيب الآلات الافتراضية، حيث يتم ترتيب الآلات الافتراضية قبل البدا بتشغيل الخوارزمية، ثم تعطي الخوارزمية جدولة تبعا للآلات الافتراضية المرتبة، وبعد ذلك وقبل كتابة الجدولة إلى ملفات Json يتم إعادتها الى ترتيبها الأصلي، كما هو موضح في الشكل:

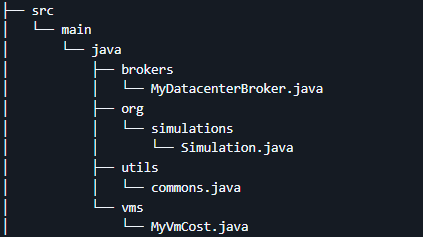


تم تنفيذ بقية التوابع الأساسية بترجمة المعادلات الرياضية المعبرة عنها بالرماز البرمجي المناسب، وبعد تنفيذ هذه التوابع:

* نُفذت في البداية الخوارزمية بهدف واحد لكل من تقليل زمن التنفيذ الكلي (makespan) وتقليل الكلفة (cost) على حدة.
* لاحقًا، تم دمج الهدفين في نسخة محسنة من الخوارزمية متعددة الأهداف (MO-LCA).
  1. تهيئة بيئة المحاكاة والتكامل مع الخوارزمية

تم إعداد بيئة المحاكاة بعد دراسة الخصائص الأساسية لبيئات الحوسبة السحابية ومحاكاة الموارد. لضبط معاملات المحاكاة بشكل مرن، تم استخدام ملفات JSON لتخزين إعدادات البيئة مثل عدد وحجم المهام، عدد الآلات الافتراضية وسرعاتها، وطرق حساب التكلفة. تقوم الخوارزمية بحفظ نتائجها (من ضمن هذه النتائج الحل الذي يمثله أفضل تشكيل فريق خلال تشغيل الخوارزمية) في ملفات JSON ليتم استهلاكها مباشرة من بيئة المحاكاة لمحاكاة الحل في البيئة السحابية.

وتكون بنية الملفات في هذه وحدة المحاكاة كالتالي:

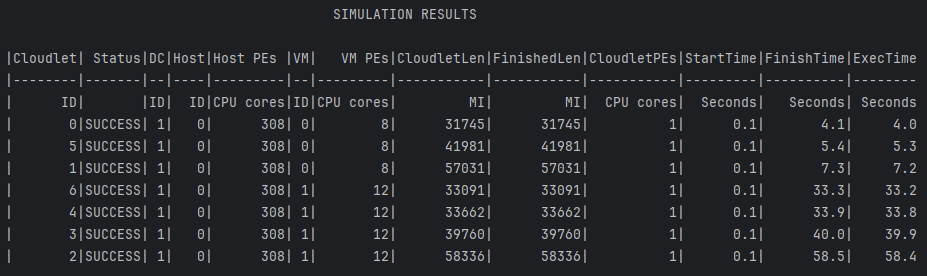


عند كتابة توابع الهدف في وحدة الخوارزمية، كان من الضروري فهم كيفية عمل بيئة التنفيذ وذلك للتنبؤ بدقة بوقت التنفيذ والكلفة حيث أن حساب قيمة الملائمة يعتمد على هتين القيمتين. يختلف وقت تنفيذ نفس الحل تبعًا لطريقة تنظيم تنفيذ المهام على الآلة الافتراضي. كما أن الكلفة تختلف حسب السياسة المعتمدة.

بعد تحليل عدة نماذج للتنفيذ (المهام المشتركة بالوقت، المهام المشتركة بالمساحة، والتنفيذ العادل كما في أنظمة Linux) تم تطوير توابع لحساب زمن التنفيذ والكلفة لكل حالة.

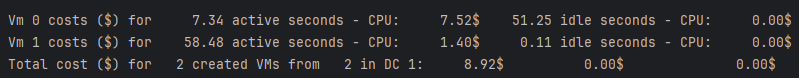
* بالنسبة لنموذج المهام المشتركة بالمساحة، تمكّنا من صياغة تابع تنبؤي دقيق نسبياً قادر على حساب زمن تنفيذ المهام استنادًا إلى خصائص الأجهزة الافتراضية وسرعة المعالجات.
* بالنسبة لنموذج المهام المشتركة بالوقت، تم إعداد تابع لحساب زمن التنفيذ، لكنه يتميز بدقة أقل نسبيًا نظرًا لتعقيد سلوك هذا النموذج.
* أما بالنسبة لنموذج التنفيذ العادل، فلم يتم اعتماده لغياب علاقة رياضية واضحة ودقيقة لتمثيله ضمن بيئة المحاكاة. بناءً على ذلك، تم تركيز التقييم النهائي على الحالتين الأولى والثانية.

وتم إجراء تجارب في بيئة المحاكاة للتعرف على آلية عمل البيئة (كيفية اسناد المهام إلى الآلات الافتراضية، كيفية حساب الكلفة وما هي الصفوف والواجهات التي نحتاج دراستها واستخدامها في التجارب وغيرها من الأسئلة المتعلقة بالمحاكاة) يظهر الشكل 10 مثال عن نتائج المحاكاة لبيئة ب 7 مهام وآلتين افتراضيتين، حيث تظهر نتائج مثل أين أسندت المهمة في أي آلة افتراضية وفي أي DataCenter، طول المهمة وقت بدأ تنفيذها ووقت انتهائها:



الشكل 13 مثال عن نتائج المحاكاة

وباستخدام الصفوف المناسبة نقوم بكتابة توابع الكلفة التي عرفناها ويظهر الشكل 11 مثال عن نتائج الكلفة في المحاكاة.



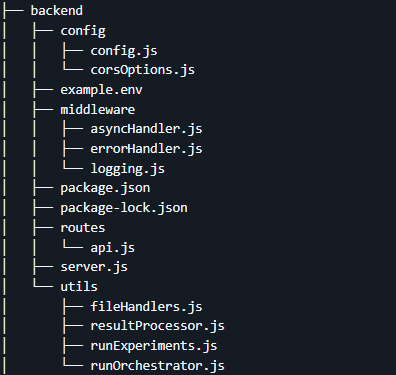
الشكل 14 مثال عن نتائج الكلفة في المحاكاة

* 1. واجهة التطبيق

تم تطوير الواجهة الخلفية باستخدام NodeJS والواجهة الأمامية باستخدام React.

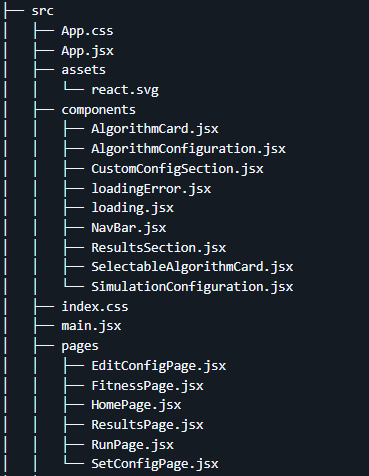
* + 1. الواجهة الخلفية (backend):

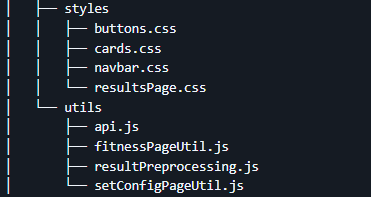
تم تطوريها ك Express.js API Layer وهي مسؤولة عن تشغيل الخوارزميات والتعامل مع ملفات النتائج ومعالجتها وفيه بنية الملفات التالية:



* + 1. الواجهة الأمامية (frontend):

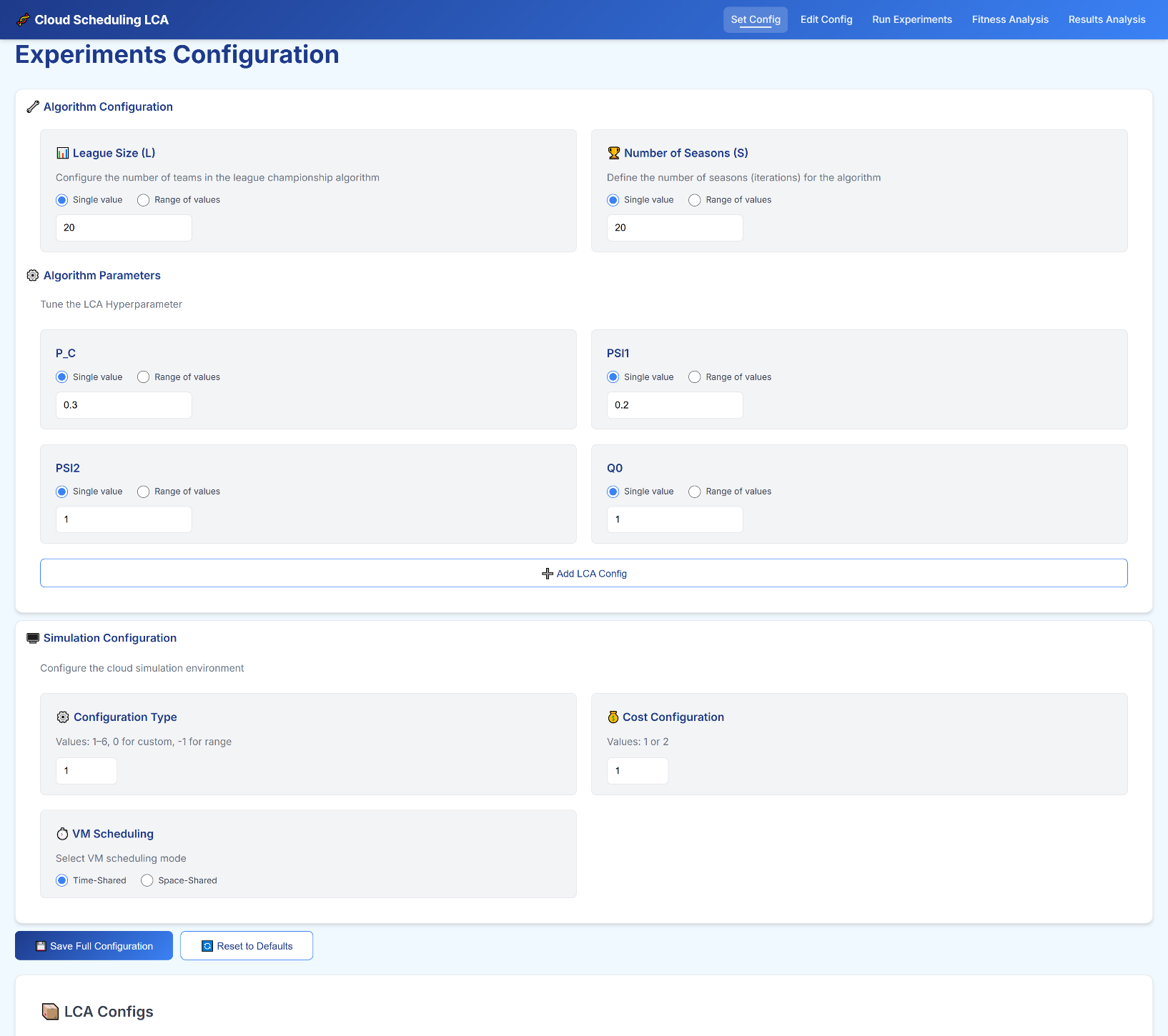
تم استخدام React SPA (Single Page Application) وفيه بنية الملفات التالية:

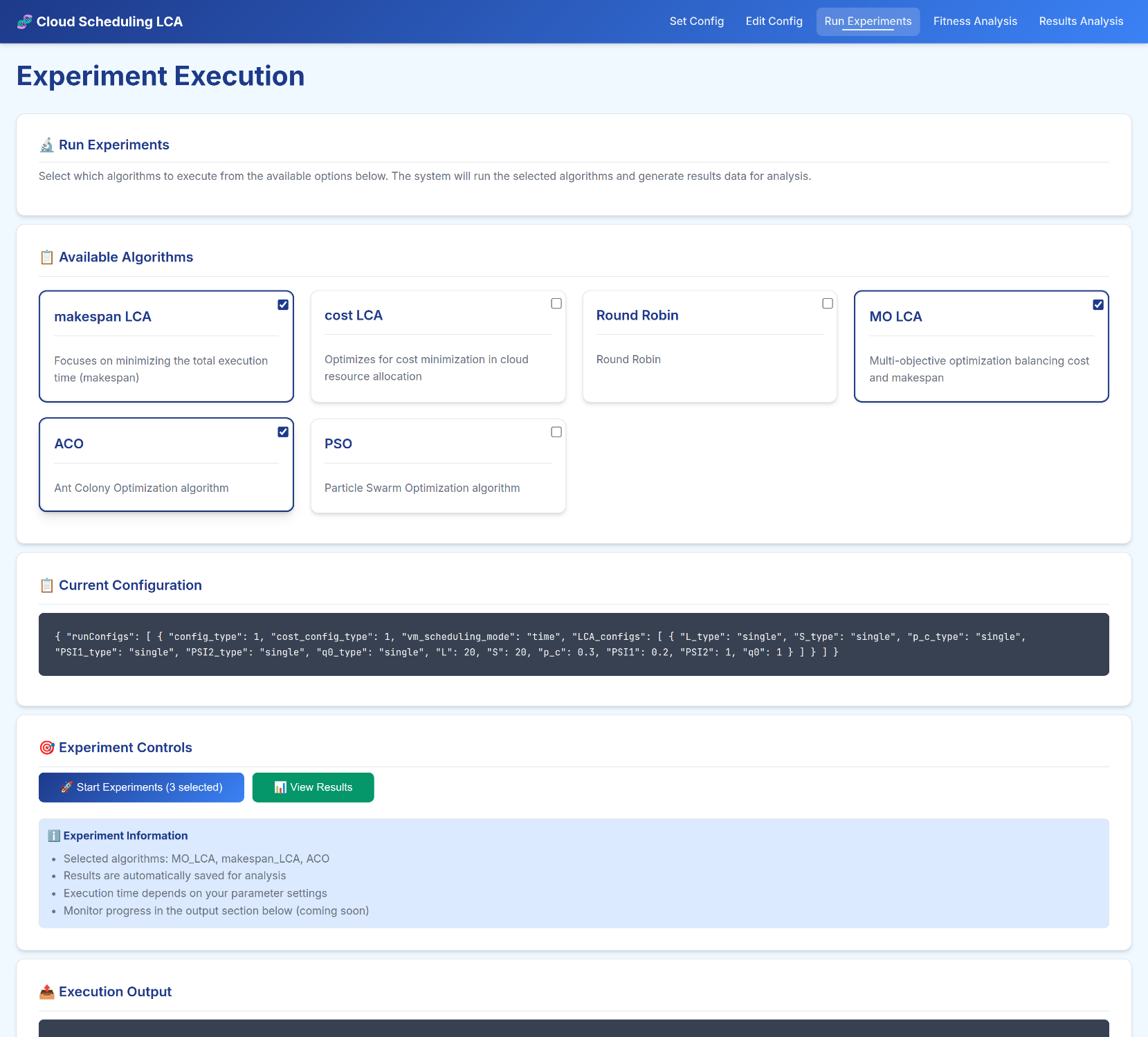


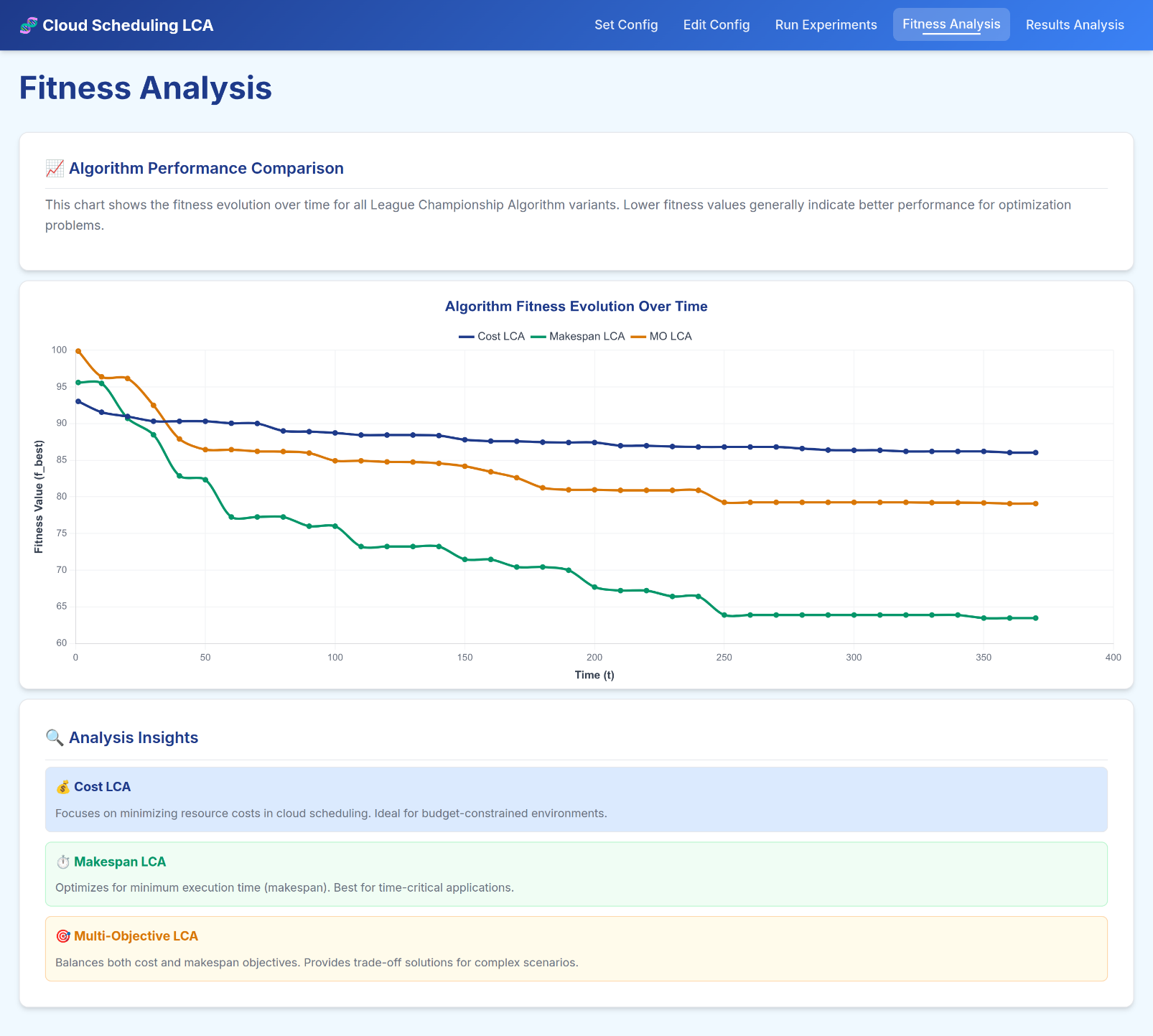


تم تطوير عدة صفحات تتيح للمستخدمين تهيئة بيئة المحاكاة ومعاملات الخوارزمية عبر واجهة رسومية سهلة الاستخدام، يمكن من خلالها اختيار الخوارزميات المراد تشغيلها للمقارنة بينها، بدء عملية الجدولة، واستعراض النتائج الناتجة بشكل مباشر. نستعرض فيما يلي الصفحات:

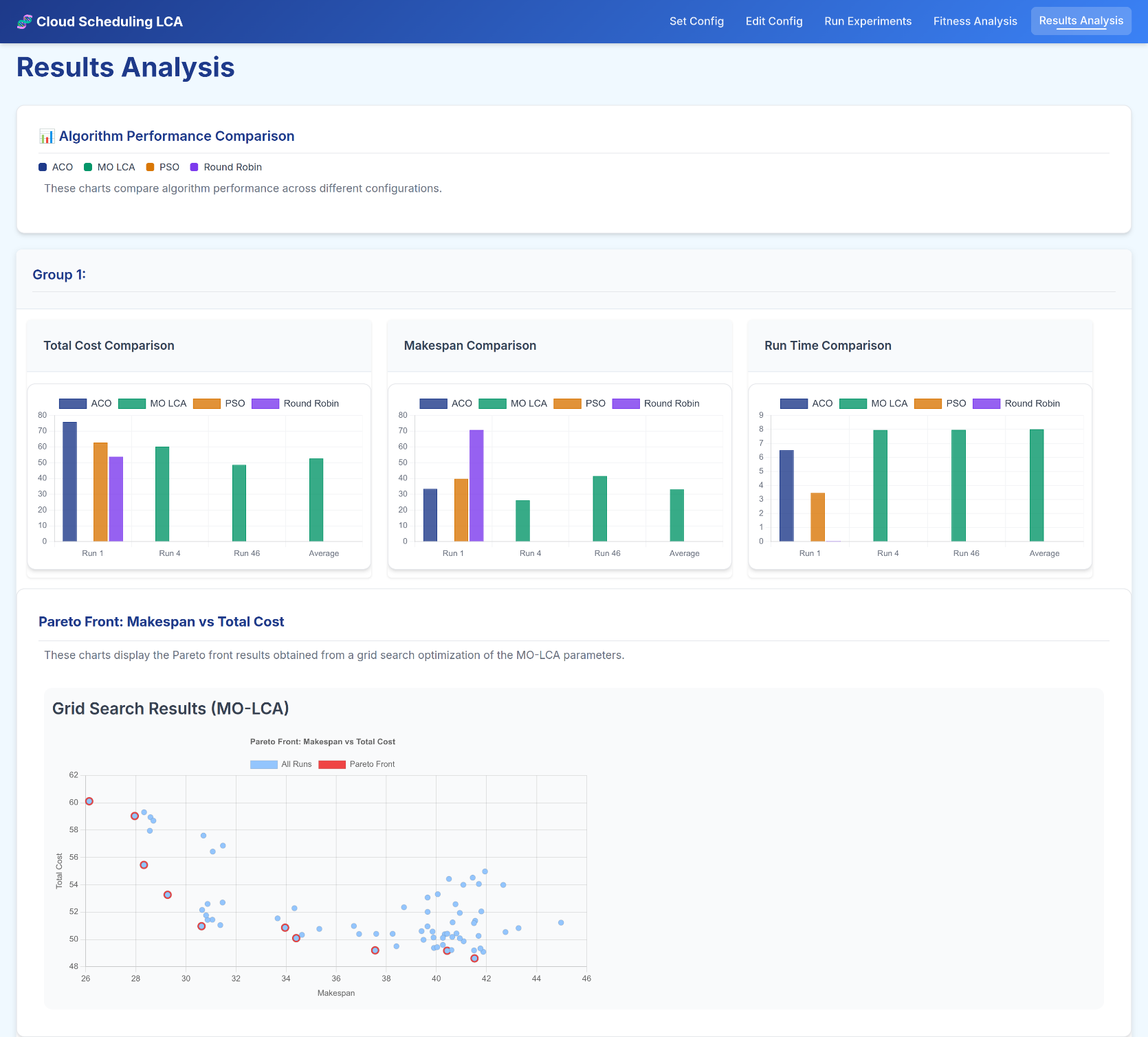
اختيار معاملات الخوارزمية والمحاكاة للتجارب:

والتي تسمح أيضا باختبار الخوارزمية على مجال من القيم لكل معال (hyperparameter tuning)

تحديد الخوارزميات المتاحة للتجارب وبدء هذه التجارب والمحاكاة:

عرض تحليل قيم تابع الملائمة:

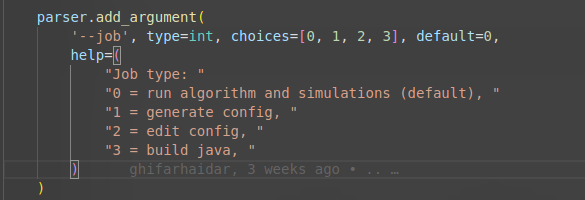
عرض نتائج المحاكاة وأداء الخوارزميات:

بالإضافة إلى مخطط يعرض نتائج ال grid search عند تشغيل الخوارزمية على مجال من القيم للHyperparameter.

* 1. واجهة التكامل

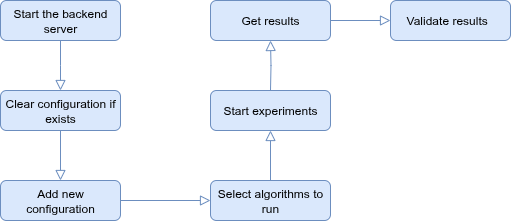
يحتوي النظام على وحدة مركزية (Broker) تلعب دور Orchestrator تقوم بالتنسيق والتحكم في تشغيل المكونات المستقلة. مما يسمح بوجود مرونة بسبب الفصل بين مكونات النظام وهذا يجعل النظام قابلا للتطوير دون التأثير على باقي الأجزاء.

وهو ملف run.py يتم استدعاءه مع معامل يحدد دور التنسيق المستدعى له:



* 1. خطة الاختبارات

تم تنفيذ طبقة اختبارات بسيطة integration tests تم فيها اختبار سير العمل الكلي للنظام:



الجدول 8 مخطط اختبار سير العمل الكلي للنظام

كما تم تنفيذ الاختبارات على github actions.

تغطي هذا الاختبار جميع ال APIs الضروري لعمل النظام واختبار تكامل الواجهة الخلفية مع وحدات المحاكاة والخوارزميات.

* 1. Logging

يدعم النظام تسجيل سجل تفصيلي (Logging) لجميع استدعاءات وطلبات APIs، بما في ذلك أوقات التنفيذ، وأي أخطاء قد تظهر أثناء التشغيل. بالإضافة إلى ذلك، يُسجل النظام نتائج تشغيل الخوارزميات وعمليات المحاكاة للبيئة السحابية بشكل مفصل في ملفات Log، حيث يتم حفظ تفاصيل مثل المُدخلات، المخرجات، والخطوات المتوسطة التي تمر بها العمليات الحسابية. مما يسمح تتبع الأخطاء وتحليلها عند حدوث المشكلات وتسمح بمراقبة أداء النظام خصوصا أداء الخوارزميات والمحاكاة والتحقق من دقة العمليات.

الفصل السابع

1. تجارب وملاحظات:

نستعرض في هذا الفصل نتائج الاختبارات وملاحظاتنا عليها.

* 1. وضع نماذج الاختبار

متغيرات البيئة السحابية التي تلعب دورا في مسألتنا هي:

عدد المهام، عدد الآلات الافتراضية، حجم المهام، عدد وحدات المعالجة في كل الة افتراضية وسرعة هذه المعالجات.

نضع عدة نماذج للبيئات السحابية لاختبار أداء الخوارزمية في بيئات مختلفة

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Num of VM | num of pes per VM | task workload (MI) | Num of tasks |  |
| 10 |  | [20K,40K] | 200 |  |
| 20 |  | [30K,60K] | 300 |  |
| 30 |  | [40K,60K] | 400 |  |
| 10 | (4,8,16) | [20K,40K] | 300 |  |
| 20 | (4,8,16) | [30K,60K] | 400 |  |
| 30 | (4,8,16) | [40K,60K] | 500 |  |

الجدول 7 أنماط المسائل

وناخذ سرعة وحدات المعالجة هي .

* 1. تجارب أولية:

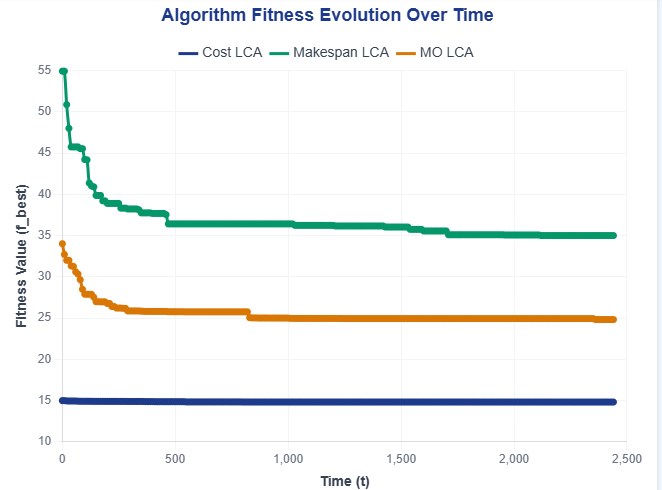
في هذه المجموعة نقوم بتشغيل الخوارزمية للتأكد أنها تعمل وأنها تحسن من قيمة توابع الملائمة، كما تم فيها اختبار عدة نماذج لتابع الكلفة، حيث تظهر المخططات التالية تحسن قيمة تابع الملائمة بمرور الوقت.

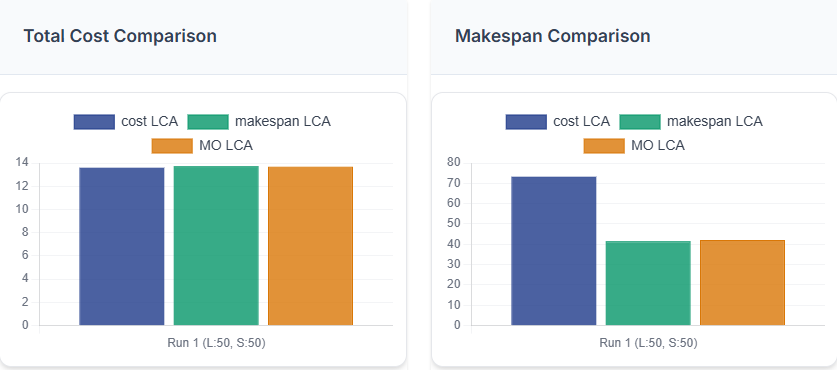
* + 1. تابع كلفة خطي

اعتمدت هذه التجارب في حساب الكلفة على الصيغة التالية ل :

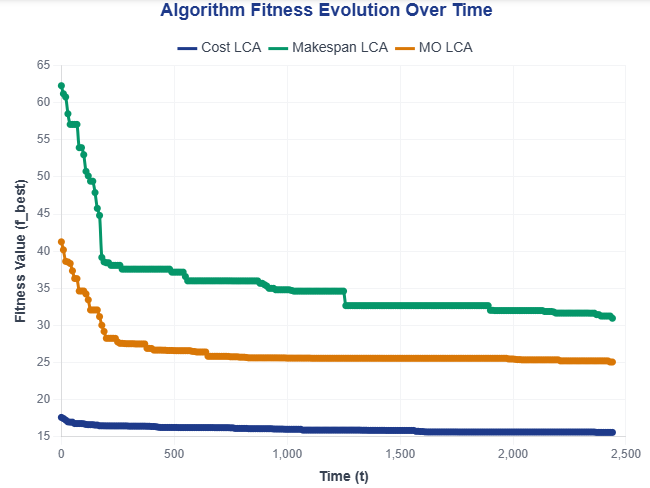
حيث هي وحدة قياس أداء الآلة الافتراضية.

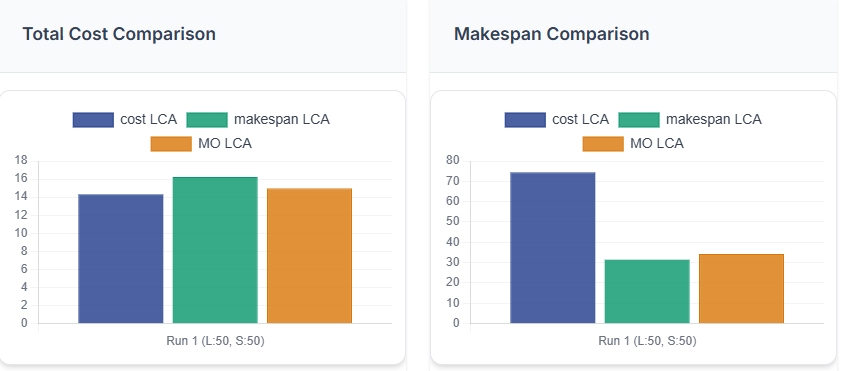
معاملات الخوارزمية





معاملات الخوارزمية





نلاحظ المشكلتين التاليتين:

لوحظ أن قيم makespan الناتجة من المحاكاة تختلف عن القيم المتوقعة عند حسابها باستخدام دالة الملاءمة في حالة time

shared، مما استدعى تحسين هذه الدالة لإعطاء حسابات أكثر دقة. حيث تم استخدام العلاقة التالية لقمية من المعادلة (14) في التجربة السابقة:

بتغيير 0.99 إلى 0.985 حصلنا على نتائج أفضل.

لوحظ أن دالة الملاءمة للكلفة لم تتغير بشكل ملحوظ، وذلك لأن النموذج يحاكي طريقة Pay-as-you-go، حيث يتم الدفع فقط عند الاستخدام، بدلاً من نموذج الحجز المسبق والدفع الثابت. في هذا النموذج، تؤدي زيادة قدرة الحوسبة لآلة افتراضية إلى زيادة الكلفة بشكل خطي.

* + 1. تابع كلفة غير خطي

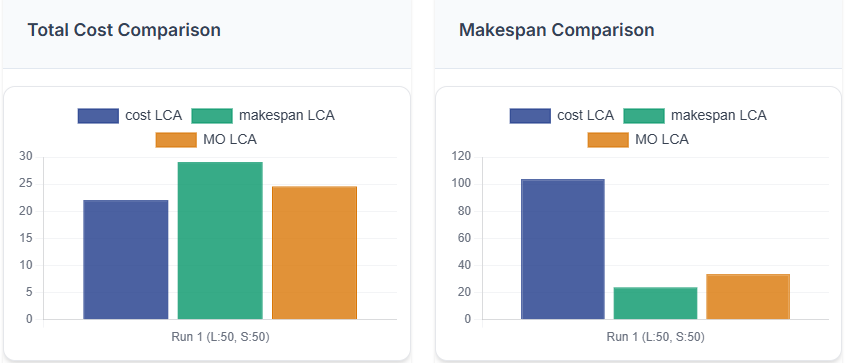
تم استخدام نموذج آخر لطريقة Pay-as-you-go، حيث تكون الزيادة في كلفة استخدام الآلة الافتراضية غير خطية:

مع .

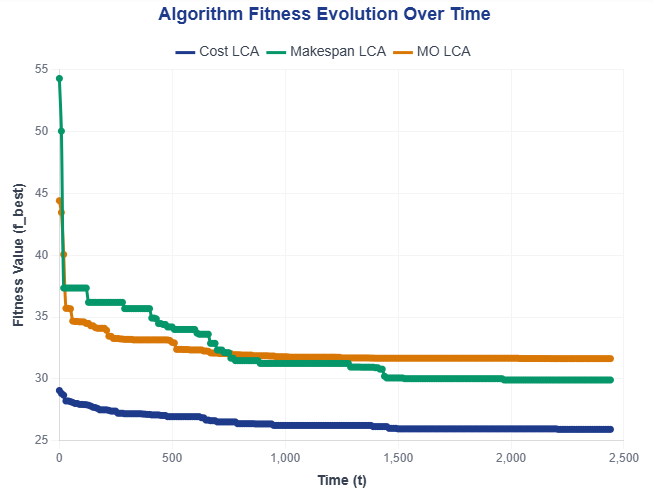
باختيار قيمة نجري التجارب التالية:

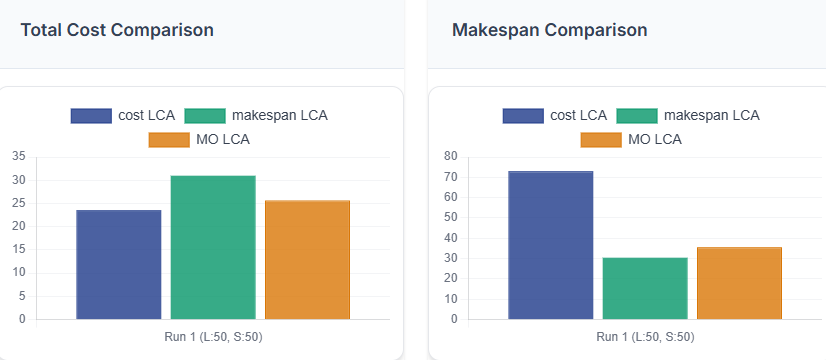
معاملات الخوارزمية





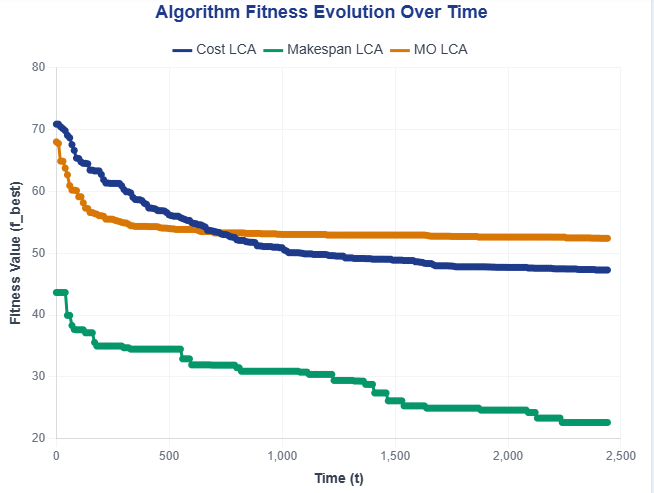
معاملات الخوارزمية

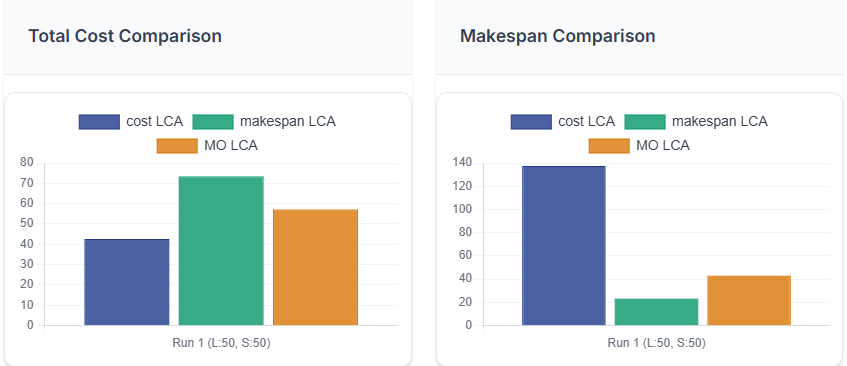




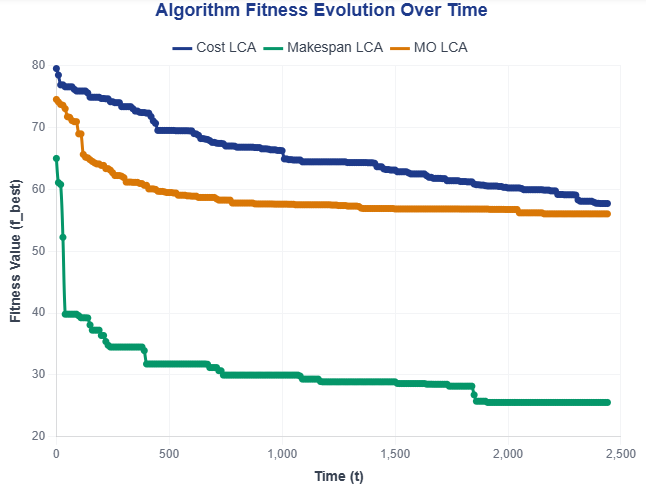
وعند اختيار

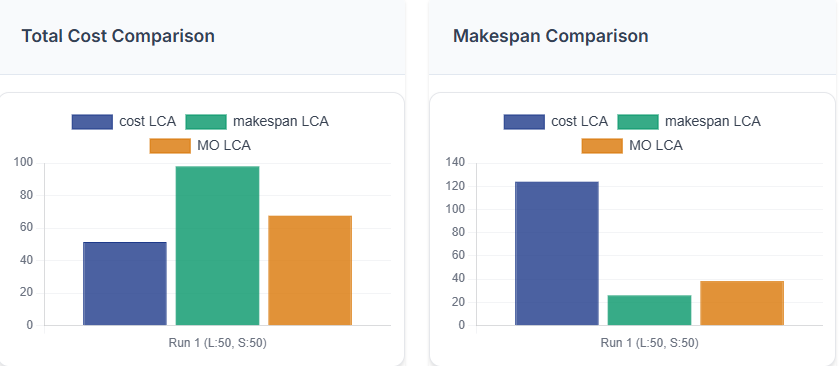
معاملات الخوارزمية





معاملات الخوارزمية



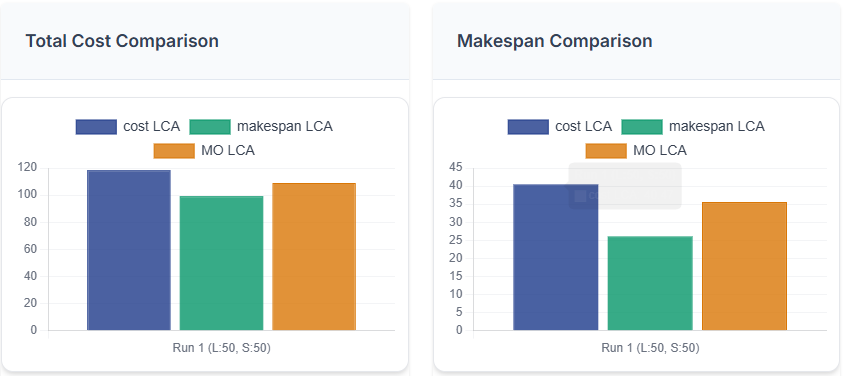


نلاحظ أن في هذه الحالة قيمة الملائمة لتابع الكلفة ليست قيمة ثابتة أو شبه ثابتة، أما في النموذج السابق كانت الكلفة ثابتة.

* + 1. تابع كلفة يراعي استهلاك الطاقة

في هذه المجموعة ناخذ بعين الاعتبار كلفة الآلة الافتراضية في حالة الخمول التي تحاكي وجود كلفة لاستهلاك الطاقة للآلة الخاملة.

نختبر في حالة space



نلاحظ أن الخوارزمية التي تحاول تقليل الكلفة تنتج كلفة أعلى من البقية، والتي تحاول تقليل زمن التنفيذ تنتج عنها كلفة أقل من البقية.

وذلك لان زيادة وقت التنفيذ تؤدي إلى ارتفاع التكلفة بسبب استمرار تشغيل الآلات لفترة أطول.

لكن في الواقع، الموارد الخاملة لا تبقى غير مستغَلة إلى وقت انتهاء جميع الآلات من مهامها حيث يمكن اسناد مهام جديدة لها، مما يقلل من هدر الموارد وكلفة الآلات الخاملة.

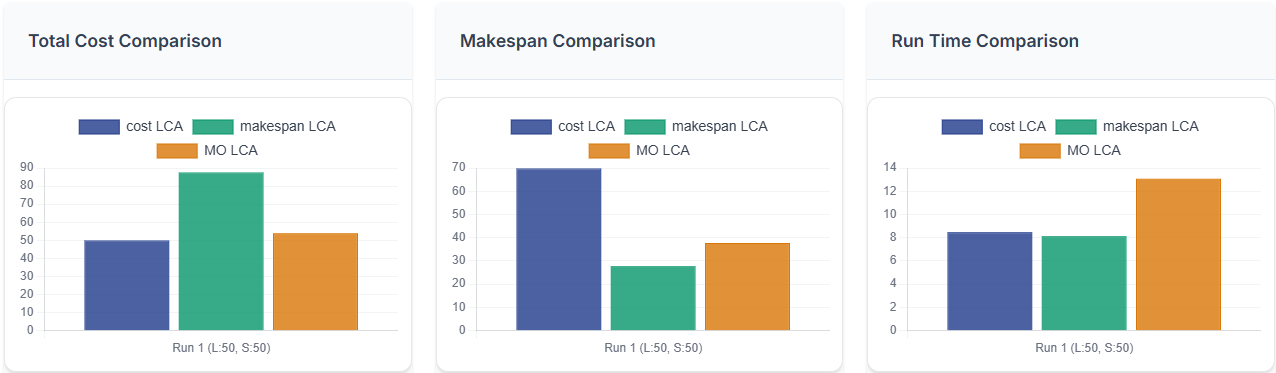
سنعتمد في التجارب اللاحقة على حساب الكلفة كما في المعادلة (21).

* 1. نتائج المحاكاة

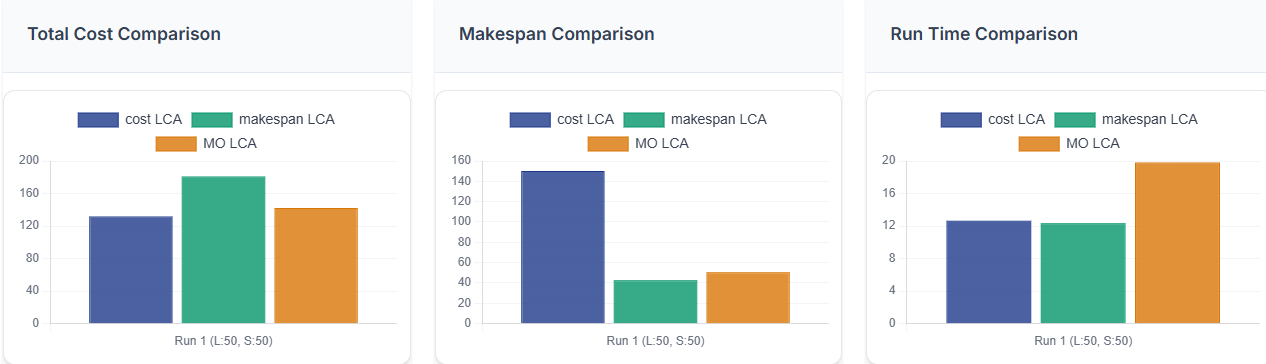
للمعاملات وباختيار

نجري تجارب محاكاة لعدد مهام والات افتراضية مختلفة، حيث تظهر المخططات التالية مقارنة لنتائج المحاكاة للجدولة الناتجة عن الحوارزميات:

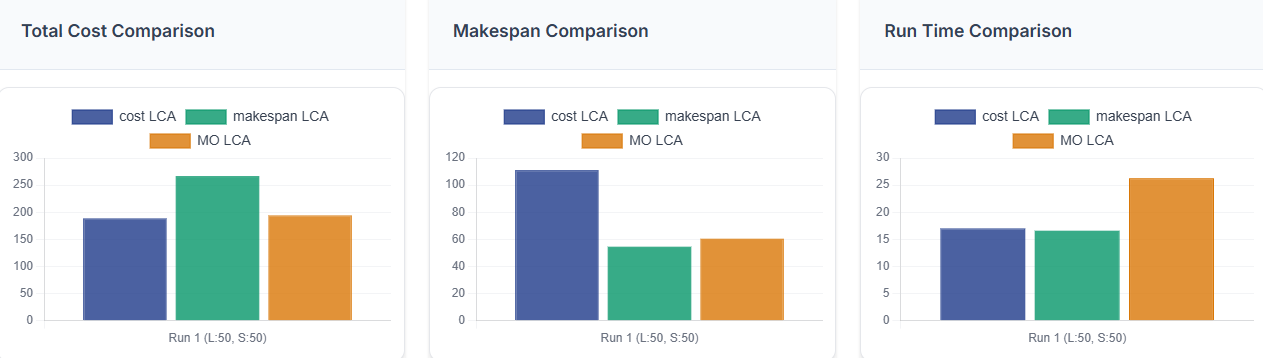
حالة المسالة p1:



حالة المسالة p2:



حالة المسالة p3:



يتبين من النتائج أن الخوارزمية متعددة الأهداف (MO LCA) تسعى لإيجاد نقطة توازن (أمثلة) بين تقليل الكلفة وتقليل زمن التنفيذ، وذلك وفقًا لقيمة التي تحدد مدى أهمية زمن التنفيذ بالنسبة للكلفة في صيغة في المعادلة (19).

نلاحظ مشكلتين: مشكلة ال scale ومشكلة وقت التنفيذ.

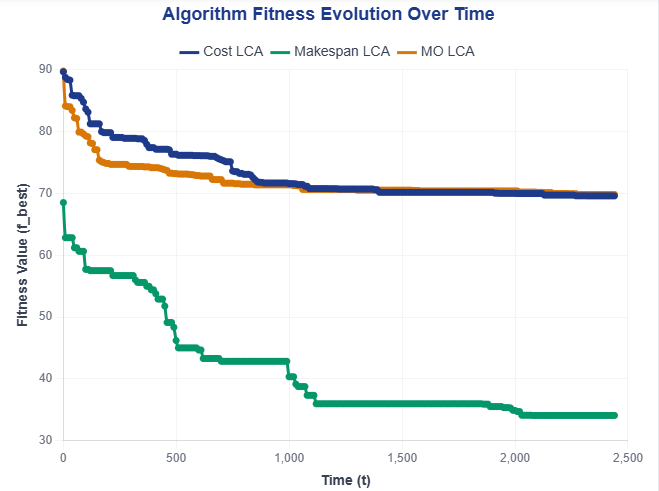
* + 1. حل مشكلة الscale

إن قيمة دالة الملائمة تختلف بين الخوارزميات، نظرا لأنها تعبر عن مقاييس مختلفة.

كما تؤدي هذه الشكلة إلى إعطاء أهمية للدالة التي لها مقياس أعلى عند حساب تابع الملائمة في المعادلة (19).

ولحل هذه المشكلة، تم إجراء عملية موازنة أو (scaling) من خلال إنشاء حل بطريقة Round Robin، ثم حساب قيمة الملائمة كنسبة مئوية من هذا الحل المرجعي.

نعيد تجربة للمعاملات وباختيار

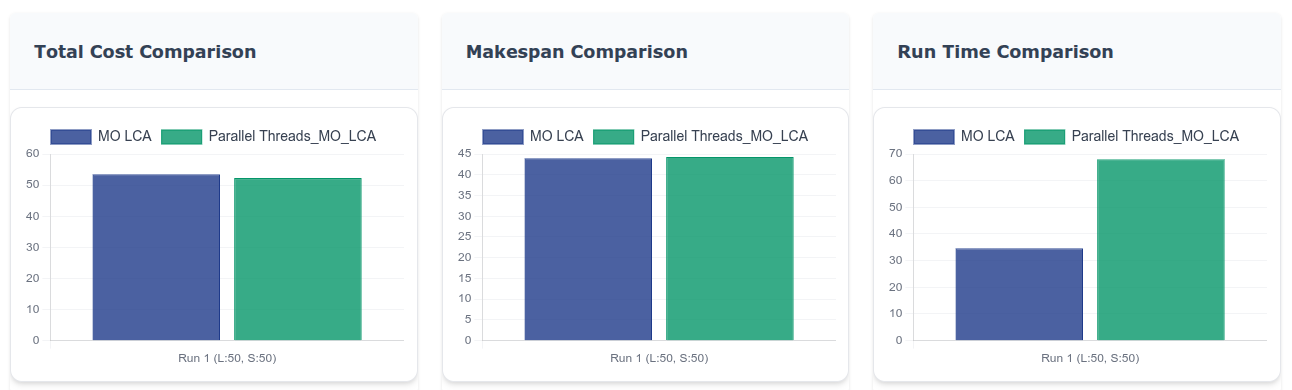


الفصل السابع

1. اختبارات اضافية

من أجل أنواع المسائل 1و2و3 التي جرى اختبارها في الفقرة ‏4.10.6-نتائج المحاكاة، نرى أن الخوارزمية متعددة الأهداف تأخذ وقتا أطول لحساب الحل من الخوارزميات بهدف واحد.

حاولنا أن ننجز الأجزاء التي تأخذ وقتا حسابيا طويلا في الخوارزمية حسابا متسلسلا باستخدام ال threads وبمقارنتها مع الأصلية:



حيث تأخذ ضعف الوقت الذي تاخذه الحسابات التسلسلية، وبعد البحث عن السبب في محاولة إصلاحه تبين أن وقت تهيئة ال threads لكل دورة يأخذ وقتا أطول من الحساب.

المراجع

1. Cloud, Hybrid. "The nist definition of cloud computing." National institute of science and technology, special publication 800.2011 (2011): 145.
2. Simmon, Eric. "DRAFT-evaluation of cloud computing services based on NIST 800-145." NIST Special Publication (SP) (2017): 500-322.
3. Bartz‐Beielstein, Thomas, et al. "Evolutionary algorithms." Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 4.3 (2014): 178-195.
4. Kashan, Ali Husseinzadeh. "League Championship Algorithm (LCA): An algorithm for global optimization inspired by sport championships." Applied Soft Computing 16 (2014): 171-200.
5. Miettinen, Kaisa. Nonlinear multiobjective optimization. Vol. 12. Springer Science & Business Media, 1999.
6. Beheshtinia, Mohammad Ali, and Amir Ghasemi. "A multi-objective and integrated model for supply chain scheduling optimization in a multi-site manufacturing system." Engineering Optimization 50.9 (2018): 1415-1433.
7. Kaur, Rajveer, and Supriya Kinger. "Enhanced genetic algorithm based task scheduling in cloud computing." International Journal of Computer Applications 101.14 (2014).
8. Kokilavani, T., and DI George Amalarethinam. "Load balanced min-min algorithm for static meta-task scheduling in grid computing." International Journal of Computer Applications 20.2 (2011): 43-49.
9. Tawfeek, Medhat A., et al. "Cloud task scheduling based on ant colony optimization." 2013 8th international conference on computer engineering & systems (ICCES). IEEE, 2013.
10. Singh, Shekhar, and Mala Kalra. "Scheduling of independent tasks in cloud computing using modified genetic algorithm." 2014 International Conference on Computational Intelligence and Communication Networks. IEEE, 2014.
11. Bhuyan, Sikha Suhani, and Ashis Kumar Mishra. "A Comparative Analysis Of Task Scheduling Algorithms Through CloudSim."
12. Mangalampalli, Sudheer, Ganesh Reddy Karri, and Utku Kose. "Multi objective trust aware task scheduling algorithm in cloud computing using whale optimization." Journal of King Saud University-Computer and Information Sciences 35.2 (2023): 791-809.
13. K. Karim, Faten, et al. "Optimizing makespan and resource utilization in cloud computing environment via evolutionary scheduling approach." PloS one 19.11 (2024): e0311814
14. Suryateja, Pericherla S. "A Comparative Analysis of Cloud Simulators." International Journal of Modern Education & Computer Science 8.4 (2016).
15. Silva Filho, Manoel C., et al. "CloudSim plus: a cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness." 2017 IFIP/IEEE symposium on integrated network and service management (IM). IEEE, 2017.
16. Sofia, A. Sathya, P. EMMANUEL NICHOLAS, and P. Ganeshkumar. "MCAMC: minimizing the cost and makespan of cloud service using non-dominated sorting genetic algorithm-II." (2019).