UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) – BarcelonaTech

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

Treball Final de Grau

Estratègies de planificació en entorns IoT-edge-Cloud

Joel Miarons Fernandez

Especialitat en enginyeria de computadors

Directora:
Rosa Maria Badia Sala
Departament:
Arquitectura de Computadors

Codirector:
Francesc-Josep Lordan Gomis

Defensat el 28/06/2022

Resum

Avui en dia, trobem dispositius intel·ligents a tot arreu. Un aspecte negatiu que tenen aquests dispositius és la poca capacitat de càlcul i d'emmagatzematge que disposen, és per això que es recolzen en el Núvol per a delegar operacions de càlcul i emmagatzematge.

Els problemes de latència entre el Núvol i els dispositius IoT han propiciat l'aparició dels conceptes Compute Contínuum com a infraestructura i Fog Computing com a paradigma de computació que proposen afegir servidors entremitjos en la infraestructura. Aquests servidors es coneixen com a servidors edge o far-edge.

El fet de tenir una infraestructura distribuïda tan complexa, i que pot canviar molt ràpidament degut als dispositius mòbils que la integren, fa que sigui més difícil explotar-la de forma eficient. Per aquest motiu, els desenvolupadors de serveis tendeixen a utilitzar models de programació (per exemple, COMPSs) que els hi simplifiquin la feina amagant els detalls de gestionar la infraestructura i, d'aquesta manera, augmentar la seva productivitat.

L'objectiu d'aquest treball és desenvolupar diferents estratègies de planificació per al model de programació de COMPSs que siguin capaços d'explotar cada una de les diferents capes de la infraestructura IoT-edge-Cloud.

Resumen

Hoy en día, hay una gran variedad de dispositivos inteligentes por todos lados. Una de las partes negativas de estos es la poca capacidad de cálculo y almacenaje del que disponen. Por este motivo se apoyan en la Nube para realizar operaciones de cálculo y almacenaje.

Al delegar tareas a la Nube hay que tener en cuenta la latencia que existen entre el dispositivo y esta. Los problemas de latencia entre la Nube y los dispositivos IoT han propiciado la aparición de los conceptos Compute Contínuum como infraestructura y Fog Computing como paradigma de computación. Estos proponen añadir servidores intermedios en la infraestructura. Los servidores que implementan estas estrategias se conocen como servidores Edge o Far-Edge.

El hecho de tener una infraestructura distribuida tan compleja (la cual puede variar muy rápidamente debido a los dispositivos móviles que la forman) hace que sea muy difícil explotarla de forma eficiente. Por este motivo, los desarrolladores de servicios tienden a usar modelos de programación que les simplifican el trabajo, escondiendo los detalles de gestión sobre la infraestructura y, de esta manera, aumentar su productividad. Uno de estos sistemas es COMPSs.

El objetivo de este trabajo es desarrollar diferentes estrategias de planificación para el modelo de programación de COMPSs que sean capaces de explotar cada una de las diferentes capas de la infraestructura IoT-Edge-Cloud.

Abstract

Nowadays, there is a vast range of smart gadgets available everywhere. One disadvantage of these gadgets is their limited computation and storage capacity. As a result, they rely on the Cloud to carry out these functions.

The delay between the device and the cloud must be considered when delegating tasks to the cloud. Due to latency issues between the Cloud and IoT devices, the ideas of Compute Continuum as an infrastructure and Fog Computing as a computing paradigm have emerged. These advocate for the addition of intermediate servers to the system. Edge or Far-Edge servers are those that employ these tactics.

It is extremely difficult to exploit such a complex distributed infrastructure (which can change very fast due to the mobile devices that comprise it). As a result, service developers seek to employ programming models that simplify their job by concealing infrastructure management details and, as a result, boost their productivity. COMPSs is one of these systems.

The goal of this study is to provide several scheduling techniques for COMPSs programming model that can take use of each of the many levels of the IoT-Edge-Cloud infrastructure.

Índex

Índex de figures

Índex de taules

1 Motivació

1.1 Context

Avui en dia la paraula IoT està molt present en la nostra vida quotidiana, tot i que aquests dispositius tenen una limitació què és la poca capacitat d'emmagatzematge i còmput que disposen. Això implica que aquests dispositius s'hagin de connectar a Internet i utilitzin el Núvol per delegar funcions de càlcul i emmagatzematge.

Les infraestructures del Núvol tendeixen a instal·lar-se en espais allunyats de la gent, i l'augment de recursos connectats al Núvol ha provocat un temps de resposta major quan els dispositius IoT han de delegar tasques al Núvol. Una solució a aquest problema és el Compute Contínuum com a infraestructura i el Fog Computing com a paradigma de computació que s'expliquen amb més detall a la secció 1.3.1.

Un altre aspecte a tenir en compte és el sistema distribuït complex que es crea entrellaçant els diferents dispositius mòbils i IoT, els servidors edge i el Núvol. En tenir tants dispositius volàtils provoca que aquests dispositius poden entrar i sortir fàcilment de la xarxa. Això comporta que fer serveis per a aquesta xarxa i explotar-la de forma eficient sigui complex. Aquest és el principal motiu que fa que els desenvolupadors de serveis es decantin per utilitzar models de programació que els simplifiquen la feina i no hagin de gestionar l'estat de la infraestructura ni els recursos disponibles. Un exemple de model de programació és COMPSs[1] que explicarem en detall a la secció 1.3.2. A més, cal destacar que en aquests models de programació una de les coses que més impacte té en el rendiment és la política de planificació que distribueix la computació entre els recursos.

Aquest document conté la memòria treball final de grau que s'ha dut a terme a la *Facultat d'Informàtica de Barcelona* (FIB), del grau d'*Enginyeria Informàtica* i l'especialitat *Enginyeria de Computadors*. El treball s'ha fet dins del grup de recerca *Workflows and Distributed Computing*, el qual centra la seva investigació en computació distribuïda. Aquest grup pertany al *Barcelona Supercomputing Center - Centro Nacional de Supercomputación* (BSC-CNS), centre especialitzat en *High Performance Computing* (HPC) i encarregat de gestionar el superordinador *MareNostrum 4*, un dels més potents d'Europa. El grup de recerca esmentat prèviament se centra, principalment, en el desenvolupament del projecte de COMPSs.

Així doncs, l'objectiu d'aquest treball consisteix a desenvolupar diferents estratègies de planifi-

cació per al model de programació COMPSs per a ser utilitzats en entorns IoT-Fog-Núvol. A més, analitzar quines d'aquestes estratègies són les més adients en cada part de la infraestructura i mesurar el rendiment d'aquests planificadors.

1.2 Actors implicats

D'entrada, tenim l'equip de desenvolupament amb la Rosa Maria Badia com a responsable del projecte que dicta els objectius i marca el ritme. Tanmateix, dins de l'equip de desenvolupament també trobem l'investigador que és l'encarregat de dur a terme el projecte i desenvolupar els planificadors. A més, els membres *sèniors* del grup duen a terme una funció de mentors per a la part tècnica i donar suport a l'investigador amb qualsevol dubte que li pugui sorgir en el desenvolupament del projecte.

D'altra banda, tenim els usuaris de COMPSs que desenvolupen aplicacions mitjançant COMPSs. Aquests es beneficiaran en el sentit que disposaran de nous tipus d'eines per a fer aplicacions.

A més, hi ha els usuaris finals que utilitzaran les aplicacions desenvolupades mitjançant COMPSs. El benefici que obtindran és que tindran a la seva disposició nous sistemes de planificació que podran fer servir en l'execució de les seves aplicacions i poder obtenir un millor rendiment utilitzant el planificador que més s'adapti al seu tipus d'aplicació i infraestructura.

Per acabar, també considerem els propietaris de grans infraestructures distribuïdes que disposaran de l'oportunitat de fer un ús més bo de la seva infraestructura si fan ús el planificador que s'adapti millor a la seva infraestructura.

1.3 Estat de l'Art

1.3.1 Infraestructura

Com s'ha esmentat prèviament, els dispositius IoT disposen d'un gran problema que és l'escassetat de recursos tant computacionals com d'emmagatzematge de dades. És per aquest motiu que molts d'aquests dispositius es complementen amb el Núvol per solucionar aquests punts mencionats. La dificultat que té el Núvol és la latència de resposta quan es comuniquen amb els dispositius IoT. Aquest és el motiu que s'utilitzen uns servidors intermedis anomenats servidors edge. L'avantatge d'afegir aquest pas intermedi entre el Núvol i els dispositius és reduir la latència i fer més eficient la computació. Aquest tipus d'infraestructura s'anomena Compute Contínuum[2] i la solució proposada és el Fog Computing. El Fog Computing defineix uns pilars bàsics per garantir les solucions als principals afers per garantir la seguretat, escalabilitat, autonomia, jerarquia... L'arquitectura de referència la va definir el *OpenFog Consortium*[3].

Per a disposar de més capacitat al nivell de IoT, se suggereix agrupar els dispositius que estan

propers entre ells. D'aquesta manera comparteixen recursos i en estar propers entre ells no hi ha tanta latència com si s'enviés al Núvol. A més, cal afegir que la idea d'aquests servidors edge és col·locar-los a una localització propera als dispositius IoT. Aprofitant millor els recursos que ofereix els servidors edge i d'aquesta manera no s'ha de desplegar un major nombre de servidors edge.

Ara que tenim definides les 3 capes de la nostra infraestructura, això ens planteja els diferents problemes de saber quan una tasca ha de canviar de nivell, és a dir, si val la pena executar-la en un dispositiu IoT o val més la pena remetre-la a un servidor edge. En existir aquesta qüestió, necessitem unes estratègies de planificació que siguin capaces de predir quan s'han de produir aquest salt i quan és més efectiu seguir al mateix nivell.

1.3.2 COMPSs

Com ja s'ha comentat anteriorment, COMPSs és un model de programació que s'encarrega de paral·lelitzar i distribuir aplicacions convertint l'execució d'unes funcions escollides per l'usuari en tasques asíncrones que s'executaran sobre la infraestructura distribuïda. Actualment, suporta aplicacions en Java, Python i C, a més afegir que el codi del runtime està fet en Java. Així mateix, les aplicacions desenvolupades amb COMPSs no tenen referències a la infraestructura a on s'executaran i d'aquesta manera fa que es pugui utilitzar aquest model de programació en qualsevol infraestructura, des de superordinadors fins al teu propi portàtil de casa.

Un altre aspecte a comentar és com aquest model de programació gestiona una aplicació. Es disposa d'un runtime que executa el codi principal i edita aquest codi reemplaçant les crides al runtime per tasques i gestiona els accessos a dades per evitar que es produeixin condicions de carrera. El planificador base de COMPSs té en compte les dependències entre les tasques i les planifica als recursos en funcions d'aquests. La manera de planificar depèn de la política escollida, actualment hi ha diverses polítiques que poden tenir en compte l'ordre de generació de tasques, la localitat de les dades o l'estat dels recursos. El runtime per defecte fa servir una arquitectura *Master-Worker* on el master planifica i coneix l'estat de tota la resta de nodes de la infraestructura (workers).

Per adaptar-se al Compute Contínuum, s'ha afegit una nova arquitectura totalment distribuïda anomenada agents que es diferencien de l'arquitectura tradicional amb el fet que els agents poden ser a la vegada masters i worker, i no necessiten conèixer l'estat de la infraestructura. És per aquest motiu, que els agents s'agrupen en colònies[4] per ajuntar recursos i reduir latència. Podem definir diferents tipus de topologies en aquestes colònies. Un exemple seria que hi hagués un agent arrel que veiés tots els altres agents com es pot observar a la figura 1.1. Aquesta topologia l'anomenem plana. Una altra topologia que es podria adoptar és la que trobem a la figura 1.2 que consisteix a generar un arbre amb els diferents agents. Per acabar, també tenim un altre tipus que seria fer una cadena d'agents que cada agent només veiés el seu successor,

aquest exemple el podem veure a la figura 1.3. Cal afegir, que quan es munta la topologia a l'agent que no és una fulla se li afegeix als recursos disponibles els recursos dels seus fills, és a dir, que si un agent fill té 2 CPUs i el seu pare en té 2 també, el pare disposa de 4 CPUs per a fer càlculs. D'aquesta manera no necessites saber la infraestructura per sota d'aquell agent només els recursos de què disposa i aquest agent ja repartirà les tasques dins la seva topologia.

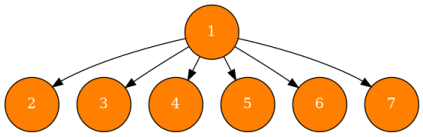


Figura 1.1: Topologia plana en agents.
Font: Elaboració pròpia

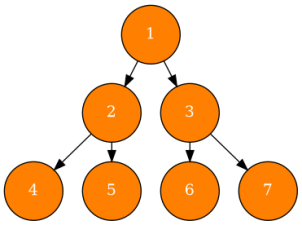


Figura 1.2: Topologia arbre en agents.
Font: Elaboració pròpia

Figura 1.3: Topologia cadena en agents.
Font: Elaboració pròpia

1.4 Justificació

Actualment, COMPSs disposa d'algunes estratègies de planificació, però que no s'adapten als entorns presentats, és per això que s'ha optat per a desenvolupar nous planificadors i fer-ne una avaluació d'aquests.

Com s'ha comentat anteriorment, disposem de diferents entorns i en ser diferents entre ells, s'ha escollit 3 estratègies de planificació cada una enfocada més en un dels entorns mencionats. Per tant, les estratègies escollides són backpressure, teoria de jocs i FIFO pels entorns IoT, edge i Núvol respectivament.

S'ha escollit una estratègia de backpressure, ja que en l'entorn IoT no es disposa d'informació completa i es necessita un planificador que es pugui adaptar als canvis de la infraestructura com expliquen Tie Qiu et al. del 2018 [5]. La segona estratègia escollida es basa en la teoria de jocs, després de llegir l'article de Sarhad Arisdakessian et al. del 2020 [6], ja que proposen una bona estratègia de planificació en entorns fog/edge i es pot aplicar a COMPSs. Per acabar, l'altra estratègia escollida consisteix a fer un *First In First Out (FIFO)* de les tasques, ja que per aquesta estratègia ens trobaríem al Núvol amb molts recursos i ens interessa que les tasques vagin ràpides per minimitzar el temps d'espera que suposa enviar al Núvol.

Les competències tècniques, que es volen assolir durant l'execució d'aquest treball, relacionades amb l'especialitat d'*Enginyeria de Computadors* són les següents:

- CEC2.1: Analitzar, avaluar, seleccionar i configurar plataformes hardware per al desenvolupament i l'execució d'aplicacions i serveis informàtics. [Una mica]
- CEC2.2: Programar considerant l'arquitectura hardware, tant en assemblador com en alt nivell. [Bastant]
- CEC2.4: Dissenyar i implementar software de sistema i de comunicacions. [En profunditat]

1.5 Organització memòria

La memòria s'ha organitzat en sis capítols, seguint l'estructura on primer es planteja la motivació, es contextualitza el treball. A continuació, es defineix el projecte indicant l'abast, metodologia a seguir el pla de treball i el pressupost. En el tercer capítol s'explica el desenvolupament dels planificadors i el disseny inicial que disposa COMPSs. Seguidament, es parla de com s'ha avaluat el prototip dissenyat utilitzant diferents tests individuals per a cada política o un global ajuntant les tres polítiques. A més, s'ha fet un informe de la sostenibilitat del projecte per veure el seu impacte ambiental, social i econòmic. Per acabar, s'han extret unes conclusions en l'àmbit tècnic, en l'àmbit de gestió del projecte i també s'ha marcat el possible treball futur.

El flux de la memòria té l'objectiu de plantejar primer el problema i una solució. A continuació, planificar el treball que es durà a terme i pressupostant-lo. Seguidament, es desenvoluparà la solució proposada i s'avaluarà el prototip. Per acabar, es presenta un informa de sostenibilitat de la solució i les conclusions extretes del treball.

2 Definició del projecte

2.1 Abast

2.1.1 Objectius

L'objectiu d'aquest projecte és millorar el rendiment de les aplicacions COMPSs en entorns del Compute Contínuum mitjançant la definició, implementació i validació de polítiques de planificació que tinguin consciència de la infraestructura. Per dur a terme aquest objectiu podem fer un desglossament de les següents tasques:

- **Model de programació de COMPSs:** adquirir els coneixements del model de programació de COMPSs en l'àmbit de sintaxi de programació i de runtime que el suporta (posant èmfasi al planificador).
- **Infraestructura:** conèixer com és la infraestructura en la qual es desenvolupa i les seves característiques.
- **Familiaritzar-se amb el planificador:** conèixer les estratègies de planificació actuals que disposa COMPSs.
- **Noves polítiques:** dissenyar i implementar noves polítiques per al model de programació COMPSs.
- **Crear un entorn de validació:** crear un entorn per a poder validar el correcte funcionament dels planificadors.
- **Avaluar el rendiment:** analitzar el comportament dels planificadors dissenyats i veure el rendiment en diferents casos d'ús.

2.1.2 Requisits

Els requisits dels planificadors que es duran a terme són els següents:

- Han de ser flexibles i s'han de poder utilitzar en la majoria de casos independentment del nombre de tasques i recursos disponibles.
- Han de ser genèrics i s'han de poder utilitzar a la majoria de recursos, que no estiguin orientats només als que s'usaran en l'àmbit de tests.

- Han de ser eficients i s'ha d'intentar que consumeixin el mínim de recursos possibles.
- Els nous planificadors han de respectar l'arquitectura del runtime de COMPSs.
- Caracteritzar els planificadors per saber quins factors influeixen en el seu rendiment i que l'usuari pugui determinar quin és el més adient per la seva aplicació i infraestructura.

2.1.3 Obstacles

Els obstacles que podem afrontar van relacionats amb els objectius fixats del treball. Un obstacle que cal remarcar seria la inexperiència de l'autor del treball, ja que no ha desenvolupat mai planificadors per a programari distribuït i no està familiaritzat amb algunes de les estratègies presentades i com funcionen. Això implicarà un esforç extra per familiaritzar-se amb aquest nou entorn i conceptes. A més, també haurà d'aprendre els conceptes relacionats amb la infraestructura, ja que tampoc els coneixia prèviament.

Un altre obstacle és el desconeixement del model de programació COMPSs. L'estudiant no coneix com funciona COMPSs i, per tant, s'haurà de familiaritzar amb el model de programació abans de començar amb el desenvolupament.

2.1.4 Riscs

En aquest apartat analitzarem els possibles riscs que podrien afectar a aquest projecte i el seu èxit.

- Alguna de les polítiques definides en l'àmbit teòric no encaixi amb l'arquitectura del runtime de COMPSs.
- Desenvolupament més lent o més difícil de l'esperat.
- Problemes amb la infraestructura que no permetin desplegar les aplicacions de test.
- Baix rendiment de les polítiques definides.

2.2 Metodologia

2.2.1 Metodologia

D'entrada, per familiaritzar-se amb l'entorn de COMPSs s'ha llegit la documentació del projecte i uns vídeos formatius per adquirir conceptes. Seguidament, s'ha paral·lelitzat una aplicació utilitzant el model de programació. A continuació, s'ha executat i analitzat l'execució per veure si s'ha paral·lelitzat correctament.

Un cop familiaritzat amb l'entorn de COMPSs, s'ha procedit amb el desenvolupament dels planificadors descrits prèviament.

La metodologia explicada a continuació s'ha aplicat per al desenvolupament dels tres planificadors esmentats anteriorment. Primer, tenim una primera fase d'investigació on es revisa l'estat de l'art sobre l'estratègia i a més, s'adquireix coneixement del funcionament d'aquesta. A continuació es passa a una fase de formulació on es planteja d'una manera abstracta, sense tenir en compte l'estructura de COMPSs, com hauria de funcionar. Un cop finalitzada aquesta fase es passa a la fase de disseny on s'adapta el resultat de la fase anterior a l'estructura de COMPSs estenent les classes del planificador base. En últim lloc, tenim la fase d'implementació on es comença a escriure codi seguint l'arquitectura plantejada en la fase anterior. Seguint aquest procés tenim la primera iteració d'un planificador, però no és ni molt menys la versió definitiva. Un cop finalitzada aquesta primera iteració es planteja si hi ha opcions de millora i es torna a repetir les fases anteriors, obviant la primera fase d'investigació i s'intenta veure si hi ha aspectes de la implementació que es poden millorar i d'aquesta manera augmentar el rendiment del planificador.

A continuació, es decideix fer tres tipus de proves per a veure com és el funcionament del planificador implementat. Primer, s'elaboren test JUnit per a comprovar que el funcionament és l'esperat. Segon, s'elabora una aplicació senzilla amb l'objectiu de mesurar mètriques en la generació i planificació de tasques per veure l'eficiència del planificador elaborat. Tercer, s'utilitza diverses aplicacions paral·lelitzades mitjançant COMPSs per a mirar com es comporta el planificador amb diferents grafs de dependències de les aplicacions.

Durant tot aquest procés, si es detecta alguna anomalia o s'observa un funcionament no esperat del planificador es torna a les fases inicials per detectar l'error i rectificar-lo per a obtenir el comportament esperat.

Finalment, quan estiguin elaborats tots els planificadors es farà un desplegament d'una infraestructura de pràctiques amb els tres entorns descrits: IoT, edge i Núvol i es provarà com es comporta la infraestructura amb els planificadors a cada un dels tres entorns i d'aquesta manera poder analitzar el rendiment que s'assoleix amb la manera que s'ha construït la infraestructura. Cal afegir que per a dur a terme els tests s'utilitzarà un portàtil Dell Latitude 7420[7] amb un processador i7-1185G7 d'11a generació amb 8 nuclis i 16 GB de ram; i l'altra plataforma on es duran a terme els tests és el superordinador MareNostrum4[8] que disposa de 48 nuclis a cada node repartits en dos processadors Intel Xeon Platinum 8160 i 96 GB de memòria ram per node i un total de 3456 nodes.

2.2.2 Seguiment

A part, s'usarà una metodologia agile[9] fent servir un kanban[10] (Trello) per a fer el seguiment de les tasques i tenir una visió global de l'estat del projecte. Tanmateix, es faran reunions setmanals, mitjançant Skype i Zoom, amb la directora i el codirector del TFG per a verificar el que es va desenvolupant i poder corregir els problemes que puguin sorgir durant el transcurs

d'aquest treball. Per altres dubtes més ràpids es faran servir eines de comunicació escrita com Slack o Skype.

Quant al desenvolupament del projecte, s'utilitzarà com a eina de control de versions el repositori de *git*[11] de COMPSs ja que és l'eina estàndard que proveeix el centre i que utilitza el grup. Aquest repositori està vinculat a un sistema de proves CI/CD[12] (Jenkins) que comprova que els canvis no alteren de forma negativa el funcionament de COMPSs. Els canvis només s'integren a la branca principal si passen una sèrie de tests predefinits. A més, per cada estratègia de planificació s'usarà una nova branca i s'intentarà a ajuntar a la principal.

2.3 Pla de treball

2.3.1 Descripció de les tasques

L'objectiu principal d'aquesta secció és fer una divisió de les tasques del projecte a més de descriure-les i mirar quines dependències tenen entre elles. Aquest treball s'ha dividit en sis grups de tasques:

- Treball previ
- Gestió del projecte
- Planificador FIFO
- Planificador backpressure
- Planificador teoria de jocs
- Avaluació global

Treball previ (TP)

- **TP.1: Vídeos instructius COMPSs.** Es visualitzen els vídeos que disposa COMPSs amb l'objectiu de familiaritzar-se amb l'entorn de treball de COMPSs i conèixer més en detall el funcionament.
- **TP.2: Documentació COMPSs.** Es llegeix la documentació que té COMPSs per al desenvolupament i execució d'aplicacions.
- **TP.3: Aplicació en COMPSs.** Es desenvolupa i s'executa una aplicació seguint el problema matemàtic de N reines i es paral·lelitza i distribueix seguint els coneixements adquirits sobre el funcionament del model de programació.
- **TP.4: Revisió estat de l'art** Es llegeixen articles científics relacionats amb la infraestructura i el model de programació per familiaritzar-se amb els conceptes.

Gestió del projecte (GP)

- **GP.1: Definició del projecte.** Es duen a terme les reunions corresponents per definir el context, abast, planificació temporal del projecte, recursos necessaris, pressupost i sostenibilitat del projecte. El resultat d'aquestes reunions es plasma en la documentació presentada al curs de Gestió de Projectes.
- **GP.2: Reunions de coordinació.** Es duen a terme reunions per a definir el projecte i coordinar el projecte durant el transcurs del projecte
- **GP.3: Documentació.** S'elabora una memòria del treball amb el desenvolupament del projecte i es documenten les diferents fases del projecte. Aquesta documentació es fa de forma paral·lela a l'execució del projecte.
- **GP.4: Presentació.** S'elabora una presentació del treball un cop finalitzada la memòria. Aquesta presentació s'utilitza per a la defensa del projecte. A més del document també es faran assajos i el guió a seguir.

Planificador FIFO (FIFO)

- **FIFO.1: Investigació.** Es revisa l'estat de l'art relacionat amb els planificadors FIFO i s'adquireixen coneixements de com funcionen aquests tipus de planificadors.
- **FIFO.2: Formulació.** Es planteja de forma abstracta quin ha de ser el funcionament del planificador.
- **FIFO.3: Disseny.** S'elabora un disseny, fent servir la formulació abstracta, tenint en compte l'arquitectura de COMPSs i quines classes s'hauran de modificar
- **FIFO.4: Implementació.** S'implementa el disseny elaborat prèviament.
- **FIFO.5: Tests unitaris.** Es fan diferents tests unitaris emprant JUnit[13] per a comprovar que el funcionament del planificador implementat és el que s'esperava.
- **FIFO.6: Mètriques.** S'usen diverses aplicacions elaborades per l'autor del treball amb la finalitat d'extreure mètriques del funcionament de planificador. Algunes d'aquestes mètriques seria el temps de creació i de planificació de les tasques depenent de certs paràmetres.
- **FIFO.7: Execució aplicacions.** S'executen aplicacions amb diferents característiques per avaluar com es comporta el planificador implementat.

Planificador backpressure (BP)

- **BP.1: Investigació.** Es revisa l'estat de l'art relacionat amb els planificadors de backpressure i s'adquireixen coneixements de com funcionen aquests tipus de planificadors.
- **BP.2: Formulació.** Es planteja de forma abstracta quin ha de ser el funcionament del planificador.
- **BP.3: Disseny.** S'elabora un disseny, utilitzant la formulació abstracta, tenint en compte l'arquitectura de COMPSs i quines classes s'hauran de modificar
- **BP.4: Implementació.** S'implementa el disseny elaborat prèviament.
- **BP.5: Tests unitaris.** Es fan diferents tests unitaris emprant JUnit per a comprovar que el funcionament del planificador implementat és el que s'esperava.
- **BP.6: Mètriques.** S'usen diverses aplicacions elaborades per l'autor del treball amb la finalitat d'extreure mètriques del funcionament de planificador. Algunes d'aquestes mètriques seria el temps de creació i de planificació de les tasques depenent de certs paràmetres.
- **BP.7: Execució aplicacions.** S'executen aplicacions amb diferents característiques per avaluar com es comporta el planificador implementat.

Planificador teoria de jocs (TJ)

- **TJ.1: Investigació.** Es revisa de l'estat de l'art relacionat amb els planificadors que fan servir teoria de jocs i s'adquireixen coneixements de com funcionen aquests tipus de planificadors.
- **TJ.2: Formulació.** Es planteja de forma abstracta quin ha de ser el funcionament del planificador.
- **TJ.3: Disseny.** S'elabora un disseny, utilitzant la formulació abstracta, tenint en compte l'arquitectura de COMPSs i quines classes s'hauran de modificar
- **TJ.4: Implementació.** S'implementa el disseny elaborat prèviament.
- **TJ.5: Tests unitaris.** Es fan diferents tests unitaris fent ús de JUnit per a comprovar que el funcionament del planificador implementat és el que s'esperava.
- **TJ.6: Mètriques.** S'empren diverses aplicacions elaborades per l'autor del treball amb la finalitat d'extreure mètriques del funcionament de planificador. Algunes d'aquestes mètriques seria el temps de creació i de planificació de les tasques depenent de certs paràmetres.

- **TJ.7: Execució aplicacions.** S'executen aplicacions amb diferents característiques per avaluar com es comporta el planificador implementat.

Avaluació global (AG)

- **AG.1: Infraestructura.** Es planteja i desplega una infraestructura amb els tres entorns descrits, IoT, edge i Núvol per a provar el rendiment dels planificadors desenvolupats.
- **AG.2: Execució aplicacions.** S'executa en la infraestructura desplegada diferents aplicacions i s'observa el comportament dels planificadors i el repartiment de tasques en la infraestructura.

Dependències entre tasques

Tot i que la tasca GP.3 depèn de la tasca AG.2 és una dependència final-final, és a dir, fins que no acabi la tasca AG.2 no es tindrà acabada la memòria, però sí que es podrà anar fent durant el transcurs del projecte. La dependència entre GP.1 i GP.3 també és de tipus final-final. Totes les altres dependències que s'estableixen en el graf són final-inici.

En la figura 2.1 podem veure les tasques amb les seves dependències. Les dependències que estan relacionades mitjançant una línia discontínua indica que són del tipus final-final mentre que les línies discontínues, indiquen dependències final-inici. Cada grup de tasques està diferenciat per colors i les dependències es mostren d'esquerra a dreta.

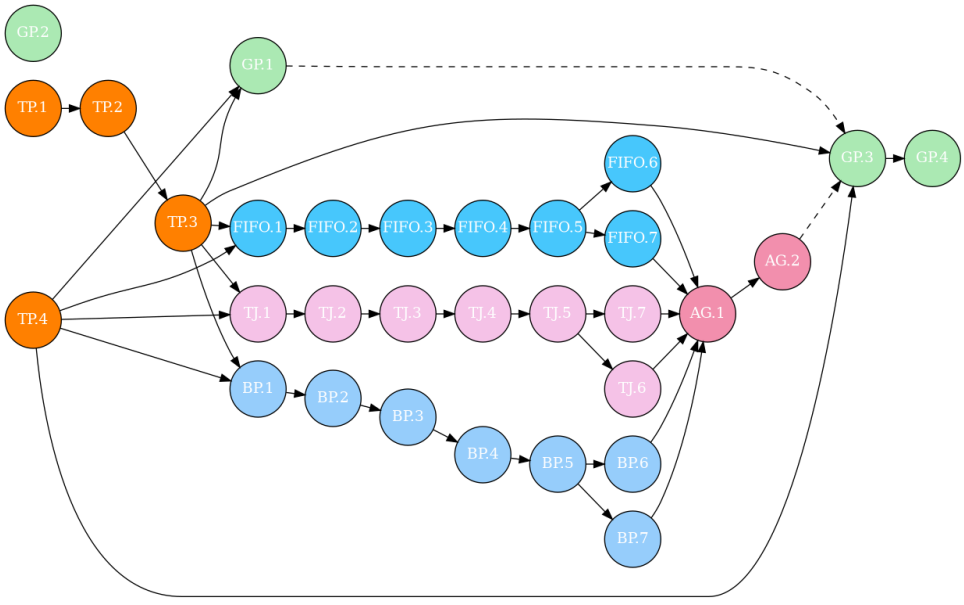


Figura 2.1: Dependències entre tasques.
Font: Elaboració pròpia

2.3.2 Recursos

Recursos humans

Els recursos humans necessaris per al desenvolupament del projecte són una cap de projecte (CP) que s'encarrega de plantejar el projecte i gestionar-lo. A més, lidera les reunions de coordinació.

Un altre recurs que necessitarem és l'Investigador sènior (IS) que s'encarregarà de donar suport a l'enginyer i a més, tindrà un rol important en les fases inicials de desenvolupament dels planificadors. Ja que està familiaritzat i domina l'entorn de COMPSs.

Per acabar, tenim l'enginyer (E) que serà l'encarregat de dur a terme el projecte, és a dir, dissenyar, desenvolupar i testar els planificadors; analitzar de forma global els planificadors a la infraestructura, i redactar la documentació.

Recursos materials

Els recursos materials que s'utilitzen per al desenvolupament del projecte són un portàtil Dell Latitude 7420 i el superordinador MareNostrum4. El superordinador s'utilitzarà sobretot per a les tasques en què es requeria treure mètriques o fer execucions d'aplicacions, mentre que el portàtil s'utilitzarà per a la gran majoria de tasques com es pot veure a la taula 2.1.

Eines de programari

Les eines de programari que s'utilitzen per al desenvolupament d'aquest projecte són les següents:

- **Overleaf.**[14] Editor de LaTeX en línia que s'utilitzarà per redactar la documentació.
- **IntelliJ.**[15] Consisteix en un editor de codi centrat en el llenguatge de programació Java que s'usarà per al desenvolupament del codi dels planificadors.
- **Zoom/Skype.**[16][17] Eines per a fer videotrucades que es faran servir per a fer les reunions de coordinació i consultes diàries.
- **Vim.**[18] Editor de text preferit per l'enginyer que s'emprarà per a elaborar programes per mesurar rendiment i per generar recursos gràfics per a la documentació, com per exemple els grafs que s'elaboren amb un llenguatge anomenat graphviz[19].
- **Trello.**[20] Eina per fer un seguiment de les tasques i l'estat en el qual es troben. S'aniran actualitzant a mesura que variï l'estat de les tasques i es comentarà el seu estat en les reunions de coordinació.
- **GitLab.**[21] Repositori de git que s'usa per al projecte de COMPSs i, per tant, s'usarà per emmagatzemar el codi dels planificadors que es desenvoluparan.
- **Ganttproject.**[22] Eina per a desenvolupar diagrames de Gantt.

- **Paraver.**[23] Eina per visualitzar traces que s'utilitzarà per fer avaluacions dels planificadors que es desenvoluparan.

2.3.3 Estimació d'esforç

Aquest treball es preveu començar-lo el dia 2 de gener de 2022 i que duri fins al dia 27 de juny de 2022 dedicant-li aproximadament 4 hores diàries els dies laborables. Es fa una previsió de dedicar-li unes 500 hores en el temps establert. El repartiment d'hores es pot veure a la taula 2.1. A més, s'ha reservat un temps entre la presentació de la memòria i l'acabament de la tasca AG.2 per si hi hagués imprevists i s'allargués alguna tasca.

Id	Tasca	Dependència	Recursos materials	RR.HH.			Eines Software
				CP	IS	E	
TP	**Treball previ**	**-**	**-**	**-**	**-**	**33**	
TP.1	Vídeos instructius COMPSs	-	Portàtil			9	
TP.2	Documentació COMPSs	TP.1	Portàtil			8	
TP.3	Aplicació en COMPSs	TP.2	Portàtil			8	IntelliJ
TP.4	Revisió estat de l'art	-	Portàtil			8	
GP	**Gestio del projecte**	-	-	-	-	155	
GP.1	Definició del projecte	TP.3, TP.4	Portàtil	4		75	Overleaf, Vim, Ganttproject
GP.2	Reunions de coordinació	-	Portàtil	20	20	20	Zoom, Skype i Trello
GP.3	Documentació	TP.3, TP.4, GP.1, AG.2	Portàtil	2	4	40	Overleaf, Vim
GP.4	Presentació	GP.3	Portàtil			20	Overleaf
FIFO	**Planificador FIFO**	**-**	**-**	**-**	**-**	**96**	**-**
FIFO.1	Investigació	TP.3, TP.4	Portàtil		4	12	
FIFO.2	Formulació	FIFO.1	Portàtil			12	
FIFO.3	Disseny	FIFO.2	Portàtil		2	12	
FIFO.4	Implementació	FIFO.3	Portàtil		4	28	IntelliJ
FIFO.5	Tests unitaris	FIFO.4	Portàtil			12	IntelliJ
FIFO.6	Mètriques	FIFO.5	Portàtil			8	Vim i Paraver
FIFO.7	Execució aplicacions	FIFO.5	MareNostrum4, portàtil			12	Vim
BP	**Planificador backpressure**	**-**	**-**	**-**	**-**	**96**	**-**
BP.1	Investigació	TP.3, TP.4	Portàtil		4	12	
BP.2	Formulació	BP.1	Portàtil			12	
BP.3	Disseny	BP.2	Portàtil		2	12	
BP.4	Implementació	BP.3	Portàtil		4	28	IntelliJ
BP.5	Tests unitaris	BP.4	Portàtil			12	IntelliJ
BP.6	Mètriques	BP.5	Portàtil			8	Vim i Paraver
BP.7	Execució aplicacions	BP.5	MareNostrum4, portàtil			12	Vim
TJ	**Planificador teoria de jocs**	**-**	**-**	**-**	**-**	**96**	**-**
TJ.1	Investigació	TP.3, TP.4	Portàtil		4	12	
TJ.2	Formulació	TJ.1	Portàtil			12	
TJ.3	Disseny	TJ.2	Portàtil		2	12	
TJ.4	Implementació	TJ.3	Portàtil		4	28	IntelliJ
TJ.5	Tests unitaris	TJ.4	Portàtil			12	IntelliJ
TJ.6	Mètriques	TJ.5	Portàtil			8	Vim i Paraver
TJ.7	Execució aplicacions	TJ.5	MareNostrum4, portàtil			12	Vim
AG	**Avaluació Global**	**-**	**-**	**-**	**-**	**24**	**-**
AG.1	Infraestructura	FIFO.6, FIFO.7, TJ.6, TJ.7, BP.6, BP.7	MareNostrum4, portàtil		4	12	
AG.2	Execució aplicacions	AG.1	MareNostrum4, portàtil			12	Vim
-	Total	-	-	26	58	500	

Taula 2.1: Taula resum de les tasques definides.
Font: Elaboració pròpia

Distribució temporal de l'esforç

Com es pot veure a la figura 2.2, s'ha elaborat un diagrama de Gantt que mostra la disposició temporal de les tasques en el temps. Aquest diagrama s'ha elaborat mitjançant el programa Ganttproject. En el diagrama es pot veure com les tasques de documentació i reunions de

seguiment es faran durant el transcurs del treball, tot i que no es faran cada dia. A més, mentre s'estigui definint el treball s'aprofitarà per a començar a estudiar l'estat de l'art de les diferents estratègies de planificació escollides. A causa de les dependències i els pocs aspectes que tenen en comú els diferents planificadors, es farà bastant seqüencial l'execució de les tasques. Tot i això, es combinarà les tasques de desenvolupar planificadors amb la redacció de la memòria.

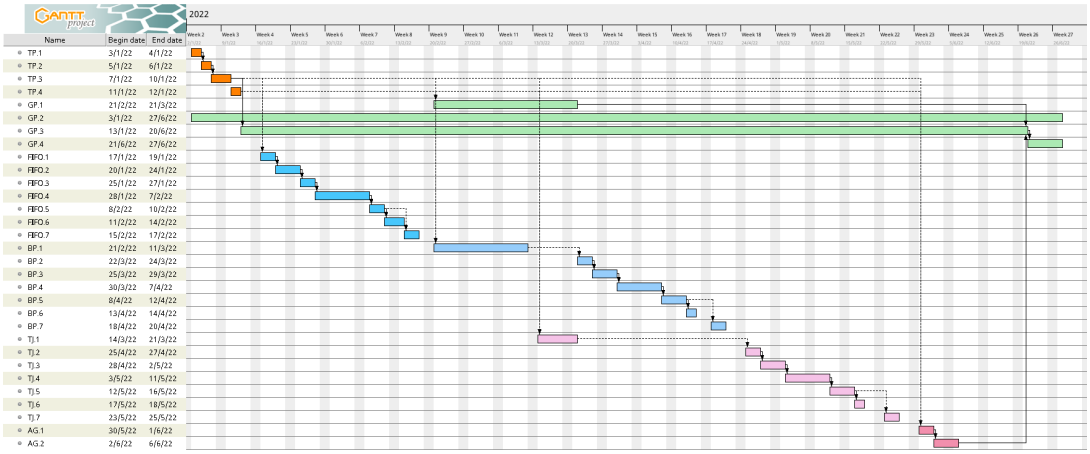


Figura 2.2: Diagrama de Gantt de les tasques proposades.
Font: Elaboració pròpia

2.4 Gestió del Risc

A continuació es detallen les mesures de contenció per a cada un dels següents riscs que es preveuen per a aquest treball: desenvolupament més lent o difícil de l'esperat, baix rendiment de les polítiques, alguna política que no encaixi a escala del runtime de COMPSs i que hi hagi problemes a la infraestructura que no permetin desplegar les aplicacions de test. Per a més informació vegeu la secció 2.1.4.

- **Alguna de les polítiques definides en l'àmbit teòric no encaixi amb l'arquitectura del runtime de COMPSs.** Això es detectaria en l'etapa de disseny i hi hauria dues opcions a seguir. La primera seria valorar si és possible retocar el runtime de COMPSs afegint el que es necessiti perquè encaixi. L'altra opció seria buscar una solució aproximada que permeti acoblar el planificador a l'arquitectura actual.
- **Desenvolupament més lent o més difícil de l'esperat.** Si ens trobéssim amb això, la solució seria afegir més recursos com per exemple demanar ajuda a altres desenvolupadors del grup de COMPSs per tal d'accelerar el desenvolupament. En cas de no poder-hi dedicar més recursos, ens veuríem obligats a reduir l'ambició de la part d'avaluació i es consideraria eliminar alguns dels tests dissenyats.

- **Problemes amb la infraestructura que no permetin desplegar les aplicacions de test.** Aquest és un risc de baixa probabilitat, ja que les diferents aplicacions que s'usaran ja han estat provades en l'entorn de desenvolupament i els únics canvis que es farien són només a la part de planificador. Tot i això, si ens trobem en aquesta situació en comptes de fer una avaluació empírica, procediríem a utilitzar un simulador per a avaluar el planificador implementat.
- **Baix rendiment de les polítiques definides.** En cas de trobar-nos en aquesta situació, s'estudiaria i es buscaria els colls d'ampolla. Un cop estiguin localitzats, es buscarien mecanismes d'optimització per tal de solucionar aquests problemes de rendiment.

2.5 Pressupost

2.5.1 Identificació de costs

El pressupost per a aquest treball s'ha dut a terme amb la definició de les tasques i esforç mencionats prèviament en aquest document (vegeu la secció 2.3.1 per la definició de tasques i la secció 2.3.3 per l'esforç). En el projecte podem classificar els costs en les següents categories:

- Personal: cap de projecte, investigador sènior i enginyer.
- Maquinari: portàtil i superordinador.
- Programari: Overleaf, Vim, GanttProject especificats a la secció 2.3.2

2.5.2 Cost de personal

Com s'ha mencionat anteriorment, es necessita una cap de projecte, un investigador sènior i un enginyer. Per consultar els salaris, s'ha consultat la web Hays [24]. Hem calculat els salaris per hora i s'han utilitzat les hores que s'han definit a la taula 2.1 i en el diagrama de Gantt de la figura 2.2. A continuació a la taula 2.2 es troba calculat el cost que tindrà el personal aplicant el 30% de la seguretat social.

Recursos humans	**Preu/hora**	**Hores**	**Cost**	**Cost SS**
Cap de projecte	30 €/h	26 h	780 €	1.014 €
Investigador sènior	23 €/h	58 h	1.334 €	1.734,2 €
Enginyer	15 €/h	500 h	7.500 €	9.750 €
TOTAL			9.614 €	12.498,2 €

Taula 2.2: Taula de costs de personal.
Font: Elaboració pròpia

2.5.3 Cost de maquinari

En aquesta secció es detallarà el cost del maquinari utilitzat per desenvolupar el treball. Els recursos que es preveuen usar són els esmentats a la secció 2.3.2

El primer element que es fa servir és un portàtil Dell Latitude 7420 [7] amb un preu de 1874,30 €.

El segon element de maquinari que se'n farà ús és el supercomputador MareNostrum4 que no es pot quantificar el seu valor, però sí que es pot fer una estimació assumint que el preu de nucli per hora és de 10 cèntims i cada node té 48 nuclis, és a dir, el preu d'un node per hora del MN4 és de *4,8 €/h*. Per tant, podem quantificar el valor del MN4 tenint en compte que s'emprarà per fer els tests dels 3 planificadors més l'avaluació global. Per a cada un de les 4 proves, es faran proves amb 2, 4, 8, 16 i 32 nodes amb 3 tipus d'aplicacions diferents, amb 2 mides de tasques diferents i 3 execucions per cada configuració per assegurar-se que no hi ha resultats no reals. A més, es preveu que cada execució tardarà com a màxim 30 minuts Això ens porta a la següent operació per a saber el cost que ens costarà utilitzar el MN4:

$$5 \times 2 \times 3 = 30 \; execucions \; per \; aplicaci$$

Per cada aplicació tenim 5 configuracions de recursos diferents, 2 mides de tasques i 3 execucions per configuració.

$$30 \times 4 = 120 \; execucions \; totals$$

Tenim 4 aplicacions diferents, una per cada planificador i l'altre l'Avaluació global.

$$\frac{2+4+8+16+32}{5} = 12,4 \; nodes/execuci$$

Tenim 5 configuracions de recursos amb 2, 4, 8, 16 i 32 nodes i podem calcular els nodes mitjans per execució.

$$120 \; exec \times 0,5 \; exec/h \times 12,4 \; nodes/exec = 744 \; nodes/h$$

Sabent que tenim 120 execucions i que cada execució dura 30 minuts, sabem que en una hora tindrem dues execucions i a més sabent el nombre de nodes per execució sabem els nodes que s'utilitzaran.

$$744 \; nodes/h \times 4,8 \; euro/h = 3571,2 \; euro$$

En conclusió, a la taula 2.3 podem trobar el cost dels recursos de maquinari.

Maquinari	**Cost total**
Portàtil	1.874,3 €
Raspberry Pi 3	143,85 €
Nvidia Jetson TX1	699,65 €
MareNostrum4	3.571,2 €
Total	6.289,0 €

Taula 2.3: Taula de costs del maquinari.
Font: Elaboració pròpia

2.5.4 Cost de programari

En aquest apartat, es quantifica el cost que tindrà el programari fet servir en l'execució d'aquest treball.

S'ha optat per a la utilització de programari gratuït, és a dir, no ha estat necessària l'adquisició de llicències per a fer ús del programari. Per tant, el cost d'aquest programari és de 0 €.

2.5.5 Contingència

Donada la magnitud del projecte, és important afegir una partida addicional, anomenada contingència, que s'emprarà només en el cas de l'aparició d'obstacles i imprevistos per a cobrir el cost d'aquests.

En aquest cas, ja que és un projecte d'investigació la possibilitat que es produeixin alteracions al pla inicial és relativament alta, es dedicarà un 20% del pressupost total com a contingència. A la taula 2.4 es pot veure el desglossament afegint la partida mencionada.

Partida	**Cost**	**Contingència**
Personal	12.498,2 €	2.499,64 €
Maquinari	6.289,0 €	1.257,8 €
Total	17.943,7 €	3.757,44 €

Taula 2.4: Pressupost amb partida de contingència.
Font: Elaboració pròpia

2.5.6 Imprevists

Anteriorment, s'ha definit els riscs que es poden produir en el transcurs del projecte (vegeu la secció 2.1.4) com s'actuarà quan se'n produeixi algun (vegeu la secció 2.4). Els únics riscs que suposarien una modificació del pressupost seria si el desenvolupament és més lent o difícil i s'hagués de demanar ajudar a altres investigadors seria pagar les hores a aquests investigadors. L'altre risc a tenir en compte és si alguna de les polítiques no encaixés bé amb el runtine i s'hagués de modificar aquest, això també implicaria remunerar les hores dedicades per part dels

investigadors sènior. La probabilitat que hi ha de modificar el runtime o es demani ajuda a algun investigador és del 40 i 20% respectivament i les hores que s'haurien de dedicar-li per cada risc són 12 hores aproximadament pel cost que té un investigador sènior l'hora (vegeu taula 2.2). A la taula 2.5 podem veure com afectaria el pressupost. Els altres riscs apareixen a la taula, però ja estan prevists dins les hores destinades a cada tasca.

Riscs	**Cost econòmic**	**Probabilitat**	**Cost %**
Desenvolupament lent o difícil	276 €	20 %	55,2 €
Modificació runtime	276 €	40 %	110,4 €
Problemes infraestructura	0 €	0 %	0 €
Baix rendiment	0 €	0 %	0 €

Taula 2.5: Cost addicional segons els riscs.
Font: Elaboració pròpia

2.5.7 Resum pressupost

A la taula 2.6 podem veure totes les partides del pressupost amb el seu valor i el cost total que tindrà el desenvolupament d'aquest treball.

Partida	**Cost**
Personal	12.498,2 €
Maquinari	6.289,0 €
Programari	0 €
Contingència	3.757,44 €
Risc	165,6 €
TOTAL	**22.710,24 €**

Taula 2.6: Cost total de l'execució del projecte.
Font: Elaboració pròpia

2.6 Seguiment i control de gestió

Pel que fa al control de gestió, s'aprofitarà les reunions setmanals de coordinació per dur a terme aquesta funció, ja que es pot controlar i corregir, si escau, de forma eficaç el desenvolupament del projecte. A més, d'aquesta manera s'evita haver d'afegir un mecanisme extra per fer aquest tipus de control.

En aquestes reunions, s'usaran els següents indicadors per fer el control dels costs i corregir les desviacions si es detecten.

1. **Desviació cost personal per tasca**:

$$(cost\ estimat - cost\ real) \times hores\ reals$$

Aquest indicador ens servirà per veure la diferència entre el preu estimat al pressupost i el preu que es pagarà a cada rol. Això ens servirà per calcular la diferència de costs de personal estimat amb el real. Aquesta desviació la podrem calcular quan es contracti el personal necessitat.

2. **Desviació realització de tasques**:

$$(hores\ estimades - hores\ reals) \times cost\ real$$

Aquest indicador ens servirà per saber les hores dedicades a cada tasca amb exactitud i comparar amb l'estimació d'hores que es va fer. Això a les reunions de coordinació es reportarà el temps dedicat a cada tasca definida a la taula 2.1 i se sabrà la diferència entre el preu que s'ha estimat i el preu real.

3. **Desviació cost de recursos**:

$$(hores\ estimades - hores\ reals) \times cost\ recurs$$

L'indicador presentat té la funció de calcular el cost real dels recursos assignats, més concret el temps en què s'està utilitzant el MareNostrum4 en comparació amb l'estimació que s'havia fet.

4. **Desviació total en la realització de tasques**:

$$cost\ estimat\ total - cost\ estimat\ real$$

Aquest indicador s'usarà per saber la desviació monetària entre el cost estimat i el cost que s'ha calculat amb les hores reals. Aquest s'utilitzarà un cop calculat l'indicador de desviació realització de tasques.

5. **Desviació total costs d'imprevists**:

$$cost\ estimat\ impreivists - cost\ real\ imprevists$$

Amb aquest indicador sabrem el cost que tindrà cada imprevist i com discerneix de la partida que s'havia destinat per a cada risc mencionat a la taula 2.5.

6. **Desviació total d'hores**:

$$hores\ estimades - hores\ reals$$

L'indicador definit s'emprarà per veure la diferència entre hores estimades i hores reals. Aquest es pot anar reportant a les reunions de coordinació i anar-se acumulant a mesura

que es van acabant les tasques.

2.7 Lleis i regulacions

Quant a lleis i regulacions, com que no s'emmagatzemen ni es tracta amb informació i dades personals relacionades amb el GDPR, no hem identificat cap llei o normativa que hagi de complir el projecte. A més, no es recullen mostres ni de teixits ni cèl·lules ni es tracta amb animals. Per acabar, cal mencionar que tampoc es tracta amb substàncies que puguin alterar el medi ambient ni la seguretat i salut de les persones i animals; ni tampoc hi ha altres aspectes ètics a tenir en compte. En resum, no hi ha cap llei que hagi de regular aquest projecte.

3 Desenvolupament del projecte

3.1 Disseny Inicial

En aquest treball, com s'ha comentat prèviament, s'implementaran tres estratègies de planificació. Per a facilitar la comprensió d'aquestes implementacions, és necessari conèixer el disseny del planificador base i com funciona aquest. Cal insistir que COMPSs utilitza un planificador base com a model per a desenvolupar les diferents estratègies de planificació que disposa. Per simplificar l'explicació ens centrarem en els dos principals esdeveniments que involucren tasques, quan arriba una nova tasca al sistema i quan s'acaba d'executar una tasca. Així mateix, veurem les parts principals del planificador i com interactuen entre elles.

Cal remarcar que entendre el funcionament del planificador base és crucial per al desenvolupament de les estratègies perquè si no s'acaba d'entendre el flux que segueix una tasca en el sistema comportarà que els planificadors que es dissenyin de forma incorrecta. Això comportaria un mal funcionament dels planificadors o fins i tot que no arribin a funcionar.

Tot el sistema de planificació es construeix sobre *Allocatable Action* (AA), accions que el runtime de COMPSs ha d'executar sobre els recursos disponibles; les tasques són un tipus d'acció. Les AA disposen d'una llista de predecessors i successors on es pot veure les dependències que té aquesta acció i, per tant, fins que les tasques que pertanyin a la llista de *Predecessors* no finalitzin la seva execució, l'AA no es podrà executar. Dins de les *Allocatable Action* podem trobar diferents tipus d'accions, però pel que fa als planificadors interessa destacar les *Execution Action*. Aquestes s'encarreguen de gestionar l'execució d'una acció, conté els paràmetres que necessita l'acció i també se li associa un *Core Element (CE)* i una implementació (vegeu A). Dins de la implementació trobem el camp anomenat *Requirements* que conté la informació dels recursos necessaris de l'acció per poder-se executar, és a dir, tenim la informació del nombre de CPUs, GPUs, memòria, etc. Algun dels altres tipus d'*Allocatable Action* s'encarreguen d'encendre, incrementar, reduir o apagar un *worker* o fins i tot d'ordenar una transferència d'arxius.

A continuació, aquestes tasques definides anteriorment entren al planificador a través del *Task Scheduler* (TS) com es pot veure a la figura 3.1. El *Task Scheduler* és la classe principal del planificador i s'encarrega de rebre les tasques i intentar-les planificar. A més d'encarregar-se de les tasques, també gestiona els workers als quals podrà planificar les tasques i els hi associa un *Resource Scheduler* (RS) a cadascun i té una llista amb tots els workers als quals els pot enviar

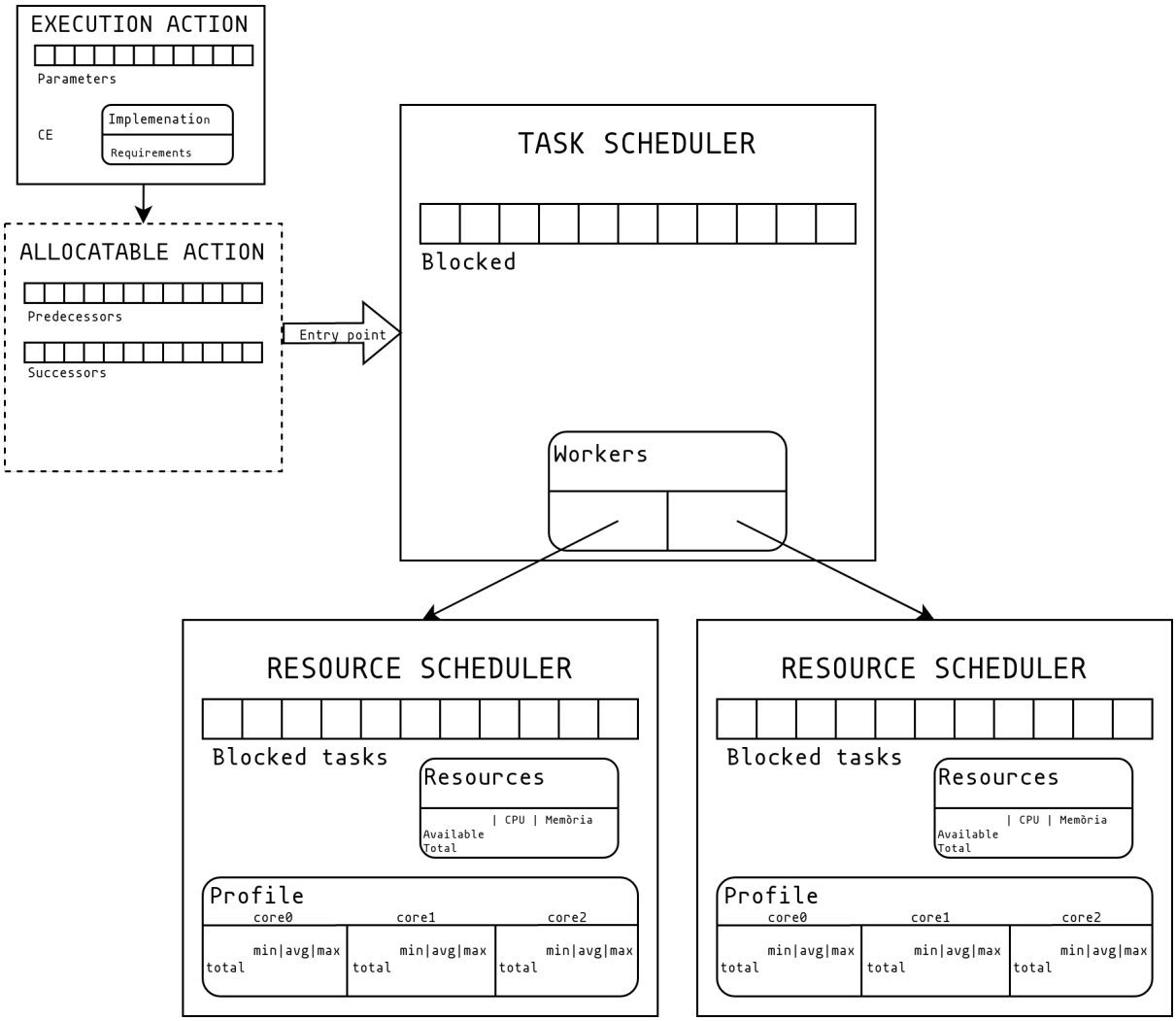


Figura 3.1: Diagrama del planificador base
Font: Elaboració pròpia

tasques. Però, tota la gestió de cada worker la fa el *Resource Scheduler* pertinent.

El *Resource Scheduler* és únic per a cada *worker* i disposa informació dels recursos que té aquell *worker* (vegeu resources a la figura 3.1) com per exemple, nombre de CPUs, memòria disponible o si disposa de GPUs i quantes en té; i també els recursos que estan lliures (*Available*), és a dir, els recursos que no estan sent utilitzats per les accions que estan executant en un moment determinat. Si arriba una acció al *worker* i els *Requirements* de l'acció són majors als recursos que trobem al *available*, llavors aquesta anirà a la cua de *blocked tasks* i es quedarà allà fins que no hi hagi suficients recursos per executar. A part de la gestió de recursos, també recull informació per cada implementació com per exemple el temps mínim o màxim que ha tardat una acció a executar i també el temps mitjà de totes les execucions així com juntament el nombre de cops que s'ha executat aquest tipus d'acció. A més, cada *Resource Scheduler* disposa d'uns

perfils per cada *CE* on es guarda informació de cada execució. Els perfils tenen diferents camps que explicarem més en detall a continuació:

- **submitTS:** instant de temps en el qual el planificador ha enviat la tasca.
- **arrivalTS:** instant de temps en el qual ha arribat la tasca al worker.
- **fetchedDataTS:** instant de temps en el qual les dades estan disponibles per dur a terme l'execució.
- **executionStart:** instant de temps en què la tasca ha començat a executar-se.
- **executionEnd:** instant de temps en què la tasca ha acabat d'executar-se.
- **endNotification:** instant temporal que el worker envia la notificació que ha acabat la tasca.
- **endTS:** instant de temps en el qual el planificador rep la notificació que ha enviat el worker.
- **executions:** nombre d'execucions que s'han fet d'un *core element*.
- **minTime:** temps mínim entre totes les execucions d'un mateix *core element*.
- **averageTime:** temps mitjà de totes les execucions.
- **maxTime:** temps màxim entre totes les execucions d'un mateix *core element*.

Cal afegir, que per defecte el temps mínim, màxim i mitjà es calculen utilitzant el *submitTS* i el *endTS*. D'aquesta manera es té en compte el temps que es tarda a enviar la tasca del master al worker, és a dir, es té en compte quin és l'estat de la xarxa tant en el moment d'enviar la tasca com en el moment de rebre-la per part del *Task Scheduler* .

Una altre qüestió a tractar és el *Score*: és una classe de suport al *Task Scheduler* per planificar les tasques. Aquesta classe té quatre camps per defecte que serveixen per comparar dues opcions de planificació quina és millor. Els camps esmentats són els següents:

- **Priority.** Indica la prioritat de la tasca.
- **ResourceScore.** La puntuació que li dona el recurs
- **WaitingScore.** El temps que s'haurà d'esperar una tasca.
- **ImplementationScore.** La puntuació que se li dona segons la implementació assignada a la tasca.

Quan s'ha de planificar una tasca, el *Score* ens ajuda a escollir quina és la millor implementació comparant les diverses implementacions que existeixen. També ajuda a escollir a quin dels

possibles recursos és millor que s'executi l'acció. A més, aquest Score es pot afegir camps o donar-li més prioritat a un camp o a un altre segons convingui a la política.

Així doncs, definits tots els components presents en el planificador base, podem entrar més en detall a com aquest gestiona quan entra una acció i quan finalitza l'execució d'una acció. Començarem analitzant com funciona quan entra una acció al sistema. El *Task Scheduler* comprova que aquesta acció no tingui cap dependència de dades; en cas que en tingui, es queda pendent de planificar-se fins que s'alliberin. Un cop alliberada, es comprova si hi ha recursos que puguin executar. En cas que no n'hi hagi cap, aquesta acció anirà a la cua de *blocked* del TS que es pot veure a la figura 3.1. Si no té dependència de dades i hi ha com a mínim un recurs disponible a on es pugui executar, el *Task Scheduler* li assigna una implementació i un *worker* a on executar-se l'acció calculant *Scores* per totes les opcions. L'altre esdeveniment important és quan finalitza una acció. En aquest cas s'alliberen les dependències de dades de l'acció que ha finalitzat i el *Task Scheduler* intenta planificar les tasques alliberades tractant-les de forma semblant a si fos una nova acció, mirant si disposa de recursos disponibles i després l'intenta planificar sempre mirant quina opció té el millor *Score.*

3.2 Planificador FIFO

3.2.1 Síntesi

La primera estratègia escollida es tracta de la política FIFO. Com el seu nom indica *First In First Out* i l'objectiu és planificar les tasques seguint l'ordre que es generen. La idea darrere aquesta estratègia és disposar d'un planificador senzill i molt ràpid per entorns distribuïts (amb tots els recursos iguals disponibles i amb disc compartit), com un superordinador. L'objectiu és que les tasques es planifiquin de pressa i no importa gaire a quin recurs. L'estratègia, com s'ha esmentat prèviament, consisteix a planificar les tasques seguint l'ordre en el qual es generen sempre que no hi hagi dependències de dades entre elles i els recursos estiguin disponibles.

Per exemple, si tenim 4 tasques numerades de l'1 al 4, la tasca 2 té una dependència de dades sobre la tasca 1 i disposem d'un recurs de 2 CPUs. La tasca 1 i 3 es planificaran al recurs i la tasca 2 i 4 quedaran bloquejades. Si finalitza la tasca 1, la dependència de dades s'allibera i, per tant, podrà entrar la tasca 2 (A). En cas contrari si acabés la tasca 3, entraria la tasca 4 al recurs (B). Aquest exemple es pot veure a la figura 3.2. En conclusió, les tasques es planificaran

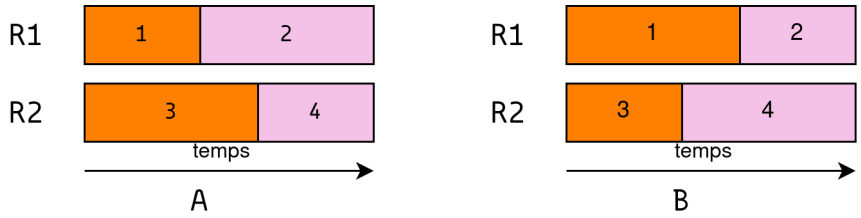


Figura 3.2: Exemple de tasques en planificador FIFO.
Font: Elaboració pròpia

seguint l'ordre en què es generen tret de què hi hagi dependències de dades entre tasques que es bloquejaran fins a resoldre aquestes dependències. A continuació, s'explicarà el disseny i la implementació que s'ha fet d'aquest planificador seguint amb el concepte presentat prèviament.

3.2.2 Disseny

Per dur a terme la implementació d'aquest planificador s'ha buscat plantejat primer fer una política base que funcioni amb un ordre que es defineix. Per aquest motiu, s'ha afegit una *PriorityQueue* per a poder ordenar les tasques amb la prioritat que defineixis. S'ha utilitzat una *PriorityQueue* perquè l'ordre pot variar, ja que hi ha tasques que no aniran a aquesta cua per dependències de dades. Aquesta cua l'anomenarem *readyQueue* com es pot veure a la figura 3.3 de color blau. També s'ha afegit els mètodes adients per accedir a la cua i modificar-la. Una altra cosa a tenir en compte, com es vol un planificador ràpid, tal com s'ha explicat a la secció anterior, s'ha optat per no tenir en compte la localitat de dades a l'hora de planificar les tasques.

Vull destacar que a la figura 3.3 veiem en blau tots els canvis que s'han fet per implementar el planificador mentre que en negre són els components del planificador base.

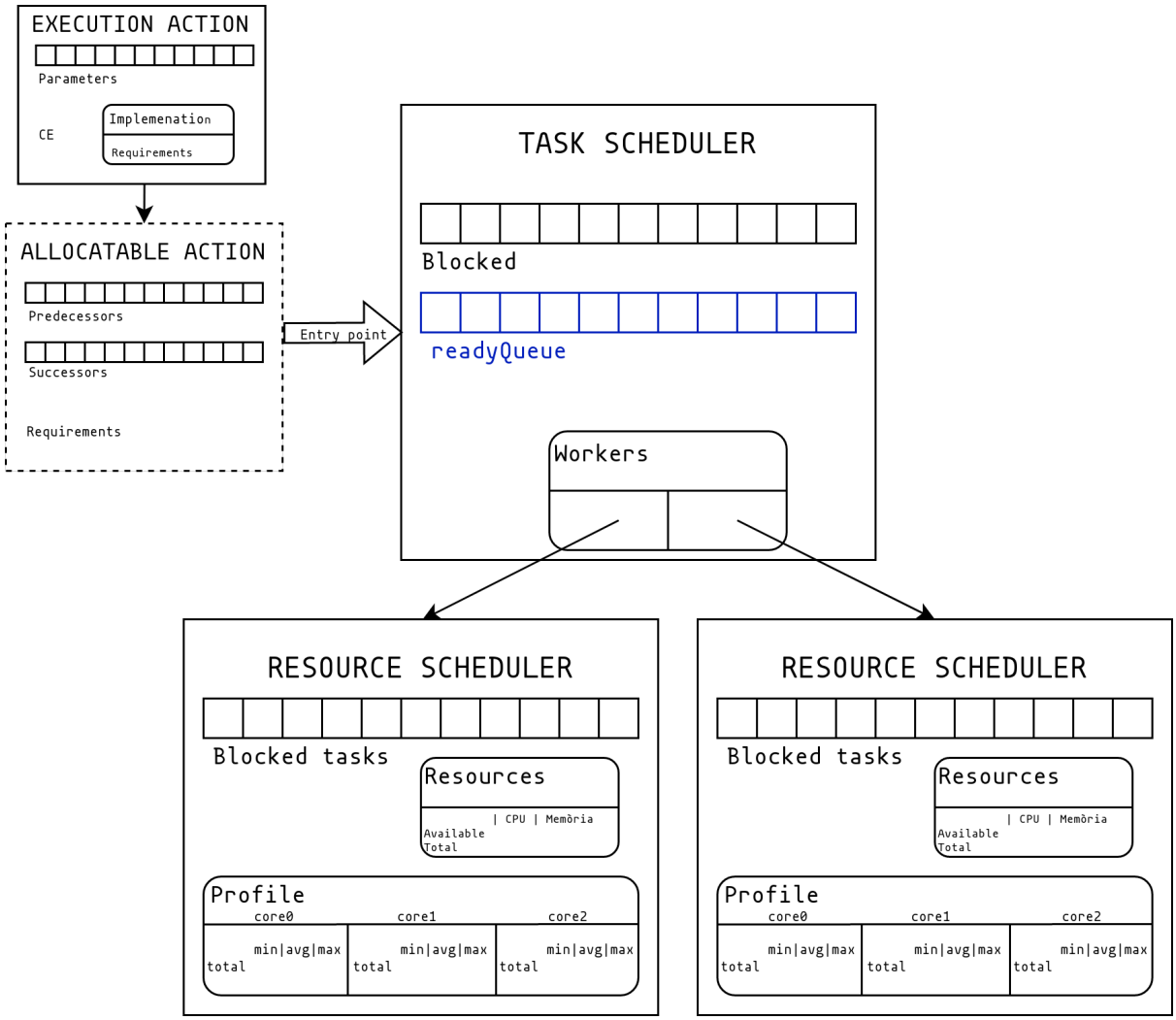


Figura 3.3: Diagrama amb els canvis fets per l'estratègia FIFO.
Font: Elaboració pròpia

S'ha modificat el funcionament del planificador quan arriba una tasca nova al sistema. En primer lloc, es comprova que la tasca que es passa com a paràmetre sigui la més prioritària de les pendents (primera de la *readyQueue*). En segon lloc, si és la tasca més prioritària, aquesta es planificarà i s'executarà. En canvi, si hi ha una altra de més prioritària, aquesta anirà a la *readyQueue*. En tercer lloc, si ens trobem en el cas que no hi ha cap recurs compatible on es pugui planificar la tasca, aquesta s'enviarà a la cua de *blocked* fins que hi hagi un recurs disponible. A la figura 3.4 podem veure l'arbre de decisió descrit quan arriba una tasca nova.

L'altre comportament que s'ha hagut de modificar del TS és quan finalitza una tasca, ja que s'han de reordenar les tasques que s'han alliberat per dependència de dades amb les tasques que

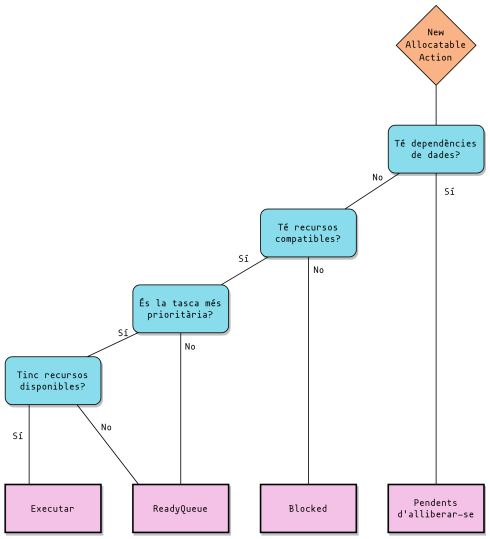


Figura 3.4: Arbre de decisió quan arriba una nova tasca.
Font: Elaboració pròpia

es troben a la *readyQueue*. El que es fa en aquest mètode es anar comprovant la primera tasca de la *readyQueue* i la cua de tasques alliberades per dependències. Quan es coneix quina és la tasca més prioritària, s'intenta planificar. Això s'anirà fent fins que una de les dues cues estigui buida o una tasca no s'hagi pogut planificar per falta de recursos. Quan ha acabat d'iterar sobre les dues cues, si la cua de tasques alliberades per dependències continua plena es fusiona aquesta cua a la *readyQueue* i s'unifiquen les tasques. A la figura 3.5 es veu de forma visual l'arbre de decisió descrit en aquest paràgraf.

Aquests canvis que s'han presentat defineixen una política base que planifica les tasques seguint un ordre definit. En el cas d'aquest treball s'ha optat de definir un ordre FIFO, tot i que es podrien optar altres tipus d'ordre com LIFO (Last In First Out) o altres criteris d'ordenació. D'aquesta manera s'ha pogut implementar un planificador molt versàtil, ja que és capaç d'adaptar-se la política que es defineixi. En aquest cas una política FIFO.

Tornant al FIFO, perquè segueixi aquesta política s'ha afegit un camp a la classe Score anomenat *actionId* que es guarda l'identificador de la tasca. Aquest nou camp l'utilitzem per comparar les tasques i com menor sigui aquest identificador més prioritat tindrà aquesta tasca. A més, per poder utilitzar aquest nou Score s'ha hagut de modificar el *RS* perquè generi aquest Score.

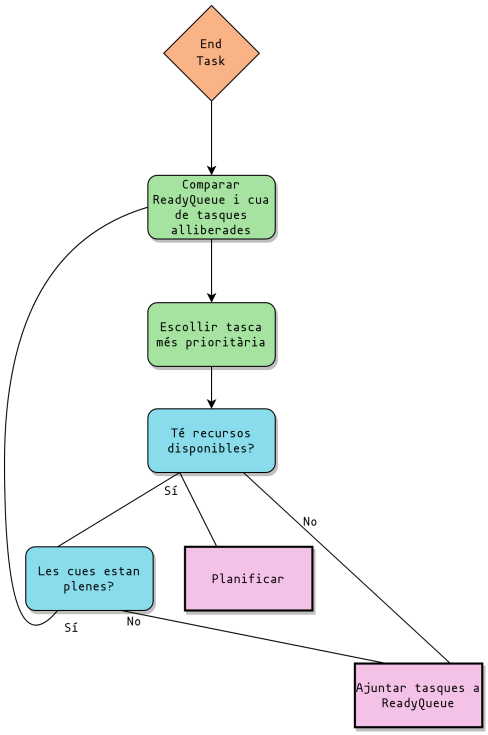


Figura 3.5: Arbre de decisió quan acaba una tasca.
Font: Elaboració pròpia

3.2.3 Validació

Per validar el funcionament del planificador s'han desenvolupat tests unitaris utilitzant JUnit. A continuació s'explicaran els diversos tests que s'han dissenyat per tal de comprovar que efectivament es manté l'ordre FIFO de les tasques en casos normals i en casos menys habituals.

Cal afegir que per dissenyar aquests tests, s'ha hagut de desenvolupar i desplegar una infraestructura de tests utilitzant classes que emula part del funcionament de la resta del runtime. També s'ha creat una classe anomenada *Validator* que s'encarrega de comprovar el funcionament del planificador, és a dir, comprova que les tasques que s'executen en el test segueixen l'ordre teòric i que quan acaba una tasca s'executa la correcta. També valida quan s'afegeixen recursos que s'escolli les tasques més prioritàries. En conclusió, s'encarrega de validar que els tests siguin correctes. A continuació, s'expliquen els diferents tests unitaris dissenyats per quan només hi ha un recurs.

1. **Un recurs i una tasca.** Aquest test és molt bàsic i serveix per comprovar que el planificador pot planificar una tasca.
2. **Validar límit de recursos.** Comprova que el planificador té en compte els recursos disponibles a l'hora de planificar les tasques i no en planifica més dels recursos que té.

A més, quan acaba una tasca i té recursos lliures planifica les tasques que han quedat bloquejades prèviament per falta de recursos. En aquest test s'ha utilitzat 4 CPUs i 5 tasques per comprovar el funcionament descrit.

3. **Comprovar prioritat sense dependències.** L'objectiu d'aquest test és comprovar que es planifiquen les tasques quan s'alliberen els recursos. Per fer això es generen 10 tasques i s'utilitza un recurs de 4 CPUs. Volem comprovar que quan acaben les 4 primeres tasques es planifiquen les 4 següents seguint l'ordre FIFO.

4. **Dependència de dades ordenades verticalment.** A la figura 3.6 es pot veure el graf de dependències que s'ha utilitzat per establir dependències de dades entre les tasques. S'utilitza un recurs de 2 CPUs, es van alliberant tasques i es comprova que es planifica la tasca amb l'identificador més baix sense dependència de dades.

5. **Dependència de dades ordenades horitzontalment.** A la figura 3.7 es pot veure el graf de dependències utilitzat, és molt similar al del test anterior, però es transposen les tasques. L'objectiu és idèntic a l'anterior test, però en aquest cas les tasques amb els identificadors més baixos es poden quedar bloquejades i executar-se tasques amb identificadors més grans sense dependències.

6. **Gestió eficient d'un recurs.** Es generen tasques que requereixen 1, 2 o 3 CPUs i s'intenten planificar seguint la figura 3.8 amb l'objectiu d'utilitzar de forma eficient el recurs tenint en compte que el planificador no ompli forats saltant-se l'ordre de les tasques, és a dir, que no intenti omplir la CPU buida que hi ha quan es planifiquen la 0 i la 1 amb la tasca 3, ja que la tasca 2 necessita 3 CPUs i només hi ha 1 de lliure. El funcionament esperat és que planifica la 0 i la 1 i quan acabin aquestes es planifica la 2 i així anar fent fins que estiguin totes planificades.

7. **Tasques bloquejades per falta de recurs.** En aquest test es vol comprovar que si es generen tasques sense tenir un recurs disponible i després s'assigna un recurs disponible, les tasques deixen d'estar bloquejades i s'executen seguint l'ordre FIFO.

Amb els tests unitaris amb un únic recurs explicats, ara és el moment d'explicar els tests que utilitzen dos recursos.

1. **Gestió eficient de dos recursos.** És molt semblant al test que s'ha explicat quan es tenia un recurs, però incrementant la mida de la infraestructura afegint un segon recurs i buscant casos on podria donar problemes com per exemple que cada recurs es quedi amb 1 CPU lliure i veure que no entren tasques que trencarien l'ordre FIFO.

2. **Dependència de dades ordenades verticalment en dos recursos.** S'usa la mateixa cadena de la figura 3.6 i es comprova que el funcionament continua sent l'esperat tenint dos recursos.

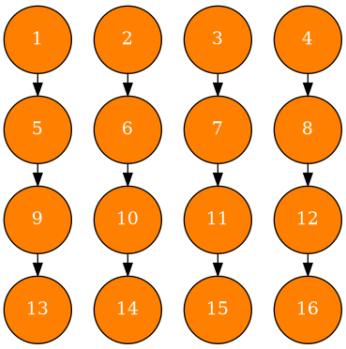


Figura 3.6: Arbre de dependències ordenades horitzontalment amb el seu identificador.
Font: Elaboració pròpia

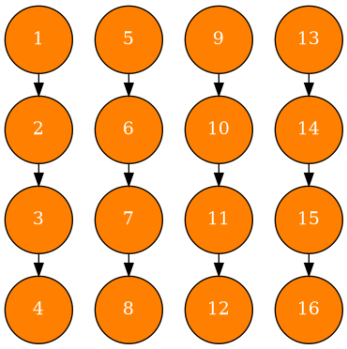


Figura 3.7: Arbre de dependències ordenades verticalment amb el seu identificador.
Font: Elaboració pròpia

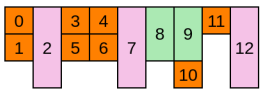


Figura 3.8: Esquema que s'ha utilitzat per planificar tasques.
Font: Elaboració pròpia

3. **Recursos amb diferent nombre de CPUs.** Es defineixen dos recursos amb un diferent nombre de CPUs, concretament s'ha utilitzat un recurs de 2 CPUs i un recurs de 3 CPUs, i tasques que necessiten 1, 2 o 3 CPUs. S'ha volgut comprovar que les tasques que requereixen 3 CPUs van al recurs que en té 3 CPUs.

4. **Comprovar que una tasca es desbloqueja quan se li afegeix un recurs compatible.** La idea és semblant al test d'un recurs on se li assigna un recurs després de generar les tasques, però la diferència és que aquest test ja té un recurs amb 2 CPUs i hi ha algunes tasques que necessiten 3 CPUs, un cop s'han generat totes les tasques es planifiquen les tasques que necessiten 1 CPU fins que arriba una que en necessita 3 i després se li afegeix el recurs per comprovar que ha sortit de la cua de bloquejades i es planifica al recurs que li pertany, és a dir, al recurs amb 3 CPUs.

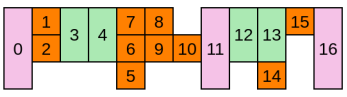


Figura 3.9: Esquema que s'ha utilitzat per planificar tasques en dos recursos.
Font: Elaboració pròpia

3.3 Planificador backpressure

3.3.1 Síntesi

La segona estratègia escollida utilitza backpressure per planificar les tasques. La idea d'aquest planificador és disposar d'una estratègia que analitzi l'estat de la infraestructura i que enviï tasques als nodes menys carregats de la infraestructura. D'aquesta manera no hi ha nodes amb moltes tasques i això provoca un alentiment de les tasques en processar-se i a més, reduir el nombre de nodes inactius sempre que sigui millor enviar tasques que executar-les en local. Aquest planificador, recull informació del temps que ha tardat una tasca en un node per intentar predir l'estat d'aquest posant-li una penalització en funció del temps que ha tardat a planificar una tasca i el temps que aquesta tasca ha estat en espera. En arribar una nova tasca pots saber l'estat de la infraestructura comprovant la penalització de cada node. En l'exemple de la figura 3.10, tenim una tasca a planificar i coneixem la penalització de cada recurs, inclòs el local. En conèixer aquesta informació, planificarem la tasca al recurs 1, ja que aquest serà el que podrà executar abans la tasca que volíem planificar.

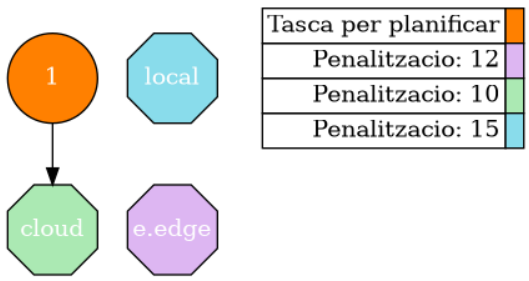


Figura 3.10: Exemple tasca en planificador Backpressure.
Font: Elaboració pròpia

En resum, l'objectiu d'aquesta estratègia és predir l'estat dels nodes i planificar la tasca al node que estigui menys carregat segons la predicció de la penalització. En la següent secció s'explicarà més en detall el disseny i implementació d'aquesta estratègia seguint el disseny esmentat al capítol 3.

3.3.2 Disseny

Aquesta estratègia de planificació s'ha implementat agafant com a base una política ja existent a COMPSs anomenada *lookahead*. De manera breu, aquest s'encarrega de mirar totes les tasques no estan planificades i les intenta planificar cada cop que finalitza una tasca. Un cop explicat la base del backpressure entrarem en el detall de les classes esteses i els mètodes que s'han modificat i afegit per implementar el funcionament descrit a la secció anterior. A la figura 3.11 podem veure de color rosat els canvis que té el *lookahead* respecte al planificador base i en blau

els canvis que s'han hagut d'afegir per poder implementar la política de backpressure.

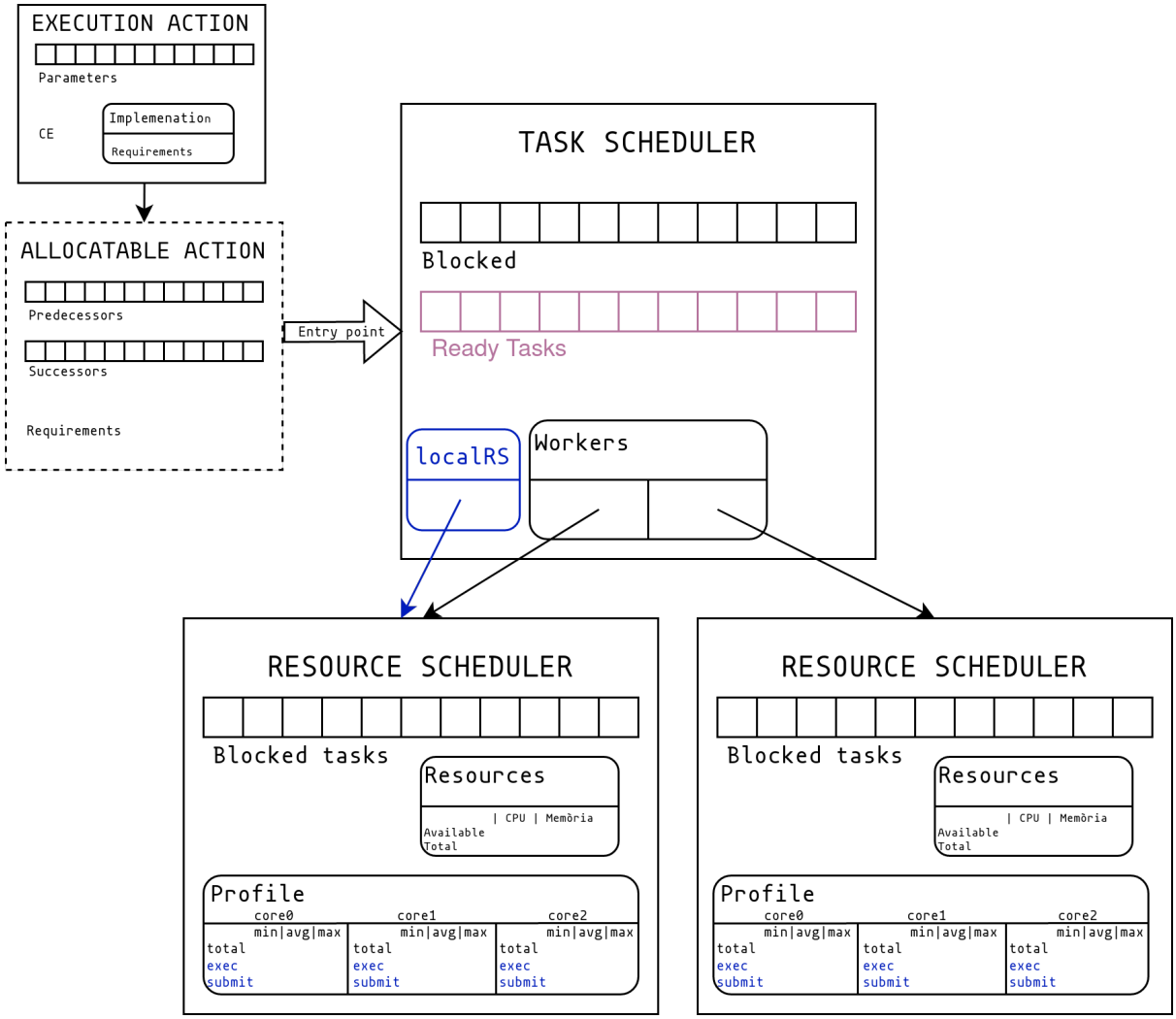


Figura 3.11: Diagrama amb els canvis fets per l'estratègia de Backpressure.
Font: Elaboració pròpia

Primerament, s'ha buscat una manera de calcular aquesta penalització i quines mètriques es tindran en compte. Una mètrica que s'ha decidit tenir en compte és el temps d'execució. A més estaria bé saber la càrrega del servidor edge, perquè pot tenir diferents dispositius IoT que li enviïn tasques que comporta no saber els recursos disponibles i el temps d'espera per executar una tasca. Amb aquestes dues mètriques se li suma una tercera que té en compte el nombre de recursos que requereix la tasca que es vol planificar, ja que com més recursos es necessitin per executar una tasca, més temps pot estar esperant a tenir-los disponibles. També s'ha decidit que el càlcul de la nova penalització es durà a terme un cop hagi finalitzat l'execució d'una tasca i, per tant, tinguem les mètriques que s'ha obtingut per l'execució d'aquesta tasca.

A l'hora d'obtenir aquestes mètriques ens hem trobat amb el problema que per saber l'estat del

servidor edge no estava implementat en el runtime cap manera d'aconseguir aquests valors. Per aquest motiu, s'ha decidit implementar aquestes mètriques al runtime, i aquesta implementació s'ha dut a terme per un investigador sènior de l'equip de COMPSs. Un cop solucionat aquest problema, s'ha fet una extensió de la classe Profile modificant la forma que es calcula el temps mitjà d'execució perquè utilitzi únicament la diferència de temps entre que una tasca es comença a executar al worker i acaba, contràriament al càlcul per defecte que s'ha explicat a la secció 3, concretament a la part del *profile*.

Per comparar dos recursos i escollir el millor s'ha decidit estendre la classe Score utilitzant el camp *waiting* per posar la penalització de cada recurs i modificar la funció de comparació perquè ho tingui en compte per escollir el millor.

A continuació, al RS se li ha afegit un mètode privat per calcular la penalització. La idea d'aquest mètode és que sigui el que defineixi la política i si es vol optar per diferents càlculs de penalització que només s'hagi de modificar aquest i que el planificador continuï funcionant. En el planificador base s'ha optat per calcular la penalització utilitzant la següent fórmula:

$$(executionStartTime - arrivalTime + oldPenalty)/2$$

Amb aquesta fórmula es vol aconseguir fer la mitjana aritmètica entre la nova penalització calculada mitjançant el *Profile* de la tasca finalitzada i la penalització actual. S'ha decidit tenir en compte la penalització anterior, ja que el càlcul serà més precís si es tenen en compte diverses instàncies de la penalització, pots fer una millor predicció i és més fàcil normalitzar els canvis bruscs de temps. D'altra banda, també s'ha decidit guardar-se l'instant de temps en què s'ha calculat l'última penalització perquè a mesura que el temps passi les tasques que estava executant el worker aniran finalitzant i si no rep noves tasques el worker hauria de disposar de recursos disponibles per a executar noves tasques. Això ens porta a la funció que genera el *Score* tenint en compte la penalització. Com hem comentat prèviament, s'utilitza el camp *waiting* per la penalització, i en aquesta funció es té en compte totes les mètriques que s'han anat presentant durant aquesta secció. Per tant, per calcular el *waitingScore* utilitzem el temps d'execució de la tasca que es vol planificar, li sumem el nombre de recursos que necessita la tasca, dividit entre la penalització ponderada pel temps transcorregut des de l'última penalització calculada.

M'agradaria afegir que per aquesta estratègia de planificació es podrien utilitzar altres maneres de calcular la penalització que siguin més precises que la que s'ha usat, fins i tot, es podria utilitzar *Machine Learning* per calcular la funció exacta que pugui calcular la penalització del worker i d'aquesta manera predir amb més precisió l'estat de cada worker. S'ha optat per aquesta manera perquè té en compte els factors que poden afectar més a l'hora de saber l'estat de la infraestructura en qüestió. Si posem tasques amb un nombre de recursos alt és més probable que s'ompli que si s'utilitzen menys recursos. Un altre factor que s'ha tingut en compte és

l'instant de temps que s'ha executat l'última tasca, d'aquesta manera com més llunyà del temps actual sigui, més probable és que el worker estigui buit, ja que hauria d'haver tingut temps per finalitzar les tasques que tenia per executar.

Per acabar amb la implementació, s'ha afegit al TS el RS local per poder-lo diferenciar del servidor edge, o servidors si en un futur es vol fer que el dispositiu IoT estigui connectat a més d'un worker. A més, s'ha modificat el mètode *scheduleAction* de forma que comprovi el Score de cada worker amb el local per mirar si és millor, i en cas afirmatiu el posa en una llista de candidats per enviar-la al mètode *schedule* de l'AA que és una altra manera d'entrar a la classe AA per poder escollir el millor worker i implementació de la tasca que es vol executar.

3.3.3 Validació

A causa de la falta de temps, s'ha optat per reduir els tests d'aquest planificador, ja que és més prioritari poder desplegar i provar un entorn com el descrit al llarg del treball amb les tres capes IoT-Edge-Núvol i provar que els planificadors funcionin i estudiar el comportament de la infraestructura. Per tant, els tests unitaris que s'han dut a terme per aquesta estratègia de planificació són dos. El primer test vol comprovar que donat dos workers a on executar una tasca agafa el worker local perquè la penalització d'aquesta és menor a la penalització del worker remot. El segon test té la intenció de fer la comprovació inversa al primer test, donat dos workers es vol comprovar que la tasca s'envia al worker remot si aquest té la penalització més baixa de les dues.

3.4 Planificador teoria de jocs

3.4.1 Síntesi

Aquesta política de planificació es basa en la teoria de jocs. Principalment, aquesta estratègia havia d'enfocar-se en els entorns edge, però després d'intentar dissenyar una estratègia que s'ajustés a aquesta capa vam veure que era massa complex i s'ajustava poc a la teoria de jocs. El problema principal que vam trobar és la manera de decidir si una tasca l'executava el servidor edge o l'enviava al Núvol. Aquesta manera de decidir es tornava massa complexa per la capacitat de càlcul que disposa el servidor edge i no acabava de ser una estratègia que reaccionava a l'estat en què es trobi el servidor sinó una sèrie de casos possibles amb la seva resposta i era simplement relacionar-los.

Un cop vam veure aquest problema, es va decidir traslladar aquesta estratègia al nivell IoT i amb un plantejament inicial es va veure que aquesta política encaixa molt millor en aquesta capa. Això fa que es pugui definir que cada dispositiu que envia al servidor edge és un jugador. En funció del ritme de producció de tasques d'aquest dispositiu s'ajusta la quantitat de tasques que cada jugador/dispositiu envia les tasques al servidor edge o s'executen de forma local. Una de les

coses que s'ha tingut en compte a l'hora de dissenyar aquest planificador és les característiques dels dispositius IoT, és a dir, la poca capacitat de càlcul que disposen. Això ens ha suposat haver de fer un planificador que no sigui complex numèricament i que qualsevol dispositiu IoT, per molt poca capacitat que disposi sigui capaç de planificar les tasques sense que afecti el seu ritme de producció.

Pel disseny d'aquest planificador, s'ha assumit les següents premisses. Les tasques només se les pot quedar en local o es poden enviar a un servidor edge (en remot); cada dispositiu genera un nombre definit i invariable de tasques; i per últim, es defineix un període on les tasques generades han de ser executades. Amb aquestes premisses hem arribat al següent disseny que consisteix a intentar maximitzar el nombre de tasques que s'envien al servidor edge sense saturar-lo. A continuació de les tasques restants, intenta maximitzar el nombre de tasques que s'executen de forma local, igual que a l'edge, sense saturar-lo i les tasques restants s'enviaran al Núvol mitjançant el servidor edge. D'aquesta manera intentem garantir que sempre es compleixi la condició del període.

A la pròxima secció s'explicarà com s'ha implementat aquest planificador.

3.4.2 Disseny

El primer pas a seguir per a desenvolupar aquest planificador ha estat definir de forma teòrica la part matemàtica d'aquest planificador, és a dir, com s'obtindrà el percentatge de tasques que s'executen a cada una de les tres capes. Es parteix que la capa IoT genera tasques a un ritme constant en períodes, és a dir, que per exemple cada minut un dispositiu IoT genera 500 tasques i un altre dispositiu genera 2000 tasques en aquest període; hem de garantir que en cada període totes les tasques s'han executat. A més, com hem comentat a la secció anterior volem maximitzar les tasques enviades al servidor edge i les restants que es reparteixin entre el Núvol i el dispositiu local en funció del cost de cada un. D'aquí podem extreure la funció a optimitzar:

$$P(s) = \sum_{i=0}^{N} (\lambda_i \times rp_i \times cl_i + \epsilon_i \times rp_i \times ce + \kappa_i \times rpi_i \times cc)$$

On tenim que N és el nombre de dispositius IoT connectats a un servidor edge; λ, ϵ i κ són el percentatge de tasques que es planifiquen en local, edge i Núvol respectivament; rp_i és el ritme de producció del dispositiu i; cl_i és el cost d'executar en local al dispositiu i; ce és el cost d'executar a l'edge tot i que assumirem que és 0 per maximitzar les tasques que van a aquesta capa; i cc és el cost d'executar al Núvol.

A part de la funció a optimitzar, també es poden extreure unes restriccions que s'han de complir que són les següents:

$$\lambda_i + \epsilon_i + \kappa_i = 1; \forall i \in N$$

Aquesta restricció ens assegura que el 100% de les tasques es planifiquen a alguna de les tres capes.

$$\left\lceil \frac{L_i}{CPU_i} \right\rceil \times T_i < P; \forall i \in N; L_i = \lambda_i \times rp_i$$

Aquesta altra restricció comprova que el temps d'executar totes les tasques planificades en local és menor al període. On tenim que CPU_i és el nombre de CPUs que té el dispositiu i, i T_i és el temps d'executar una tasca en el dispositiu i.

$$\left\lceil \frac{\sum_{N}^{i=0} E_i}{CPU_E} \right\rceil \times T_E < P; E_i = \epsilon_i \times rp_i$$

Aquesta última restricció s'assegura que el temps d'executar les tasques que envien tots els dispositius IoT a l'edge és menor al període marcat. On tenim que CPU_E és el nombre de CPUs que té el servidor edge, i T_E és el temps d'executar una tasca en el servidor edge.

Un cop definides la part matemàtica, és moment de passar a la implementació que està separada en dues parts. La primera consisteix a desenvolupar un planificador que donat el percentatge de tasques que s'executen en local i el percentatge de tasques que s'envien a l'edge sigui capaç de planificar i distribuir correctament aquestes tasques. La segona part consisteix a plantejar i desenvolupar el problema matemàtic i fer un solucionador algebraic per trobar aquest percentatge mencionat anteriorment. Aquest solucionador s'ha plantejat que per a l'àmbit del treball s'utilitzarà abans d'executar el programa per trobar el percentatge ideal tot i que de cara al futur estaria bé integrar-ho al runtime i que aquest pugui anar recalculant el percentatge per adaptar-se als possibles canvis que es puguin produir durant l'execució.

Quant a la primera part, s'ha utilitzat el planificador base explicat a la secció 3.1. Com s'ha comentat es vol un planificador que consumeixi pocs recursos de càlcul. Per això s'ha desenvolupat un planificador que rep un percentatge que són les tasques que ha d'executar en local i les va repartint entre el servidor edge (remot) i ell mateix. Per fer aquest repartiment de tasques es defineix un comptador per cada tasca que es vol planificar i es fa el mòdul del comptador amb el 100% i es mira si és major o menor que el percentatge establert inicialment. Si és major s'intenta enviar al servidor edge, mentre que si és inferior s'intenta executar de forma local. A més, s'ha establert un mètode de precaució que consisteix a enviar la tasca al node contrari al que estava previst en cas que es produeixi un error, és a dir, que si una tasca s'havia planificat per executar en local i falla s'enviarà al servidor edge i viceversa. A continuació, quan es decideix a on es planifica la tasca s'incrementa el comptador amb el valor del percentatge. Quan finalitza una tasca i s'alliberen les dependències, se segueix el mateix procediment descrit per quan arriba una nova tasca. Per finalitzar, en tenir únicament dos recursos a on poder planificar tasques, s'identifica cada recurs quan es genera amb les etiquetes de *localRS* si es tracta del recurs local, i *remoteRS* si es tracta del recurs corresponent al servidor edge. A la figura 3.12 es pot veure els canvis afegits en color blau respecte al planificador inicial de la secció 3.1.

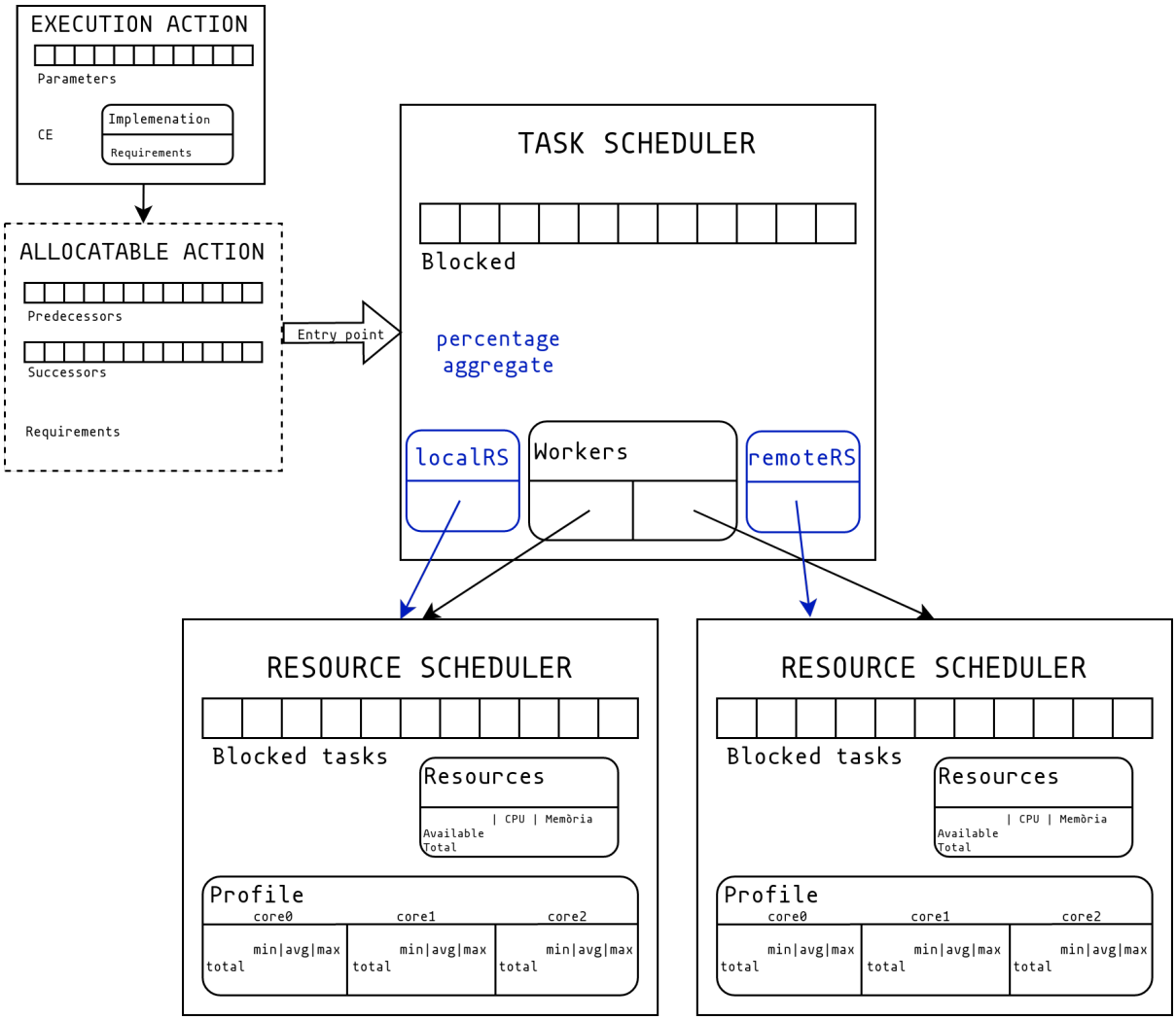


Figura 3.12: Diagrama amb els canvis fets per l'estratègia de Teoria de Jocs.
Font: Elaboració pròpia

Quan a la segona part, per desenvolupar el solucionador algebraic s'ha fet una recerca per buscar diferents solucionadors ja implementats a Java i s'ha escollit finalment utilitzar el *SimplexSolver* de la llibreria *Commons Math* d'Apache[25]. S'ha optat per un símplex, ja que és un algorisme emprat per a fer optimitzacions, com en el nostre cas que es volia maximitzar el nombre de tasques, i un dels requisits consisteix a ser àlgebra lineal i com hem vist anteriorment aquest requisit també es compleix. A més, tenim una funció a optimitzar i un seguit de restriccions definides prèviament. Un cop definit el solucionador que es vol utilitzar, s'ha procedit a fer el desenvolupament d'aquest convertint les equacions vistes prèviament al format especificat per la llibreria.

Durant el desenvolupament ha aparegut una dificultat: el nombre de dispositius IoT que es troben a la capa més baixa de l'estructura és un nombre no definit que varia depenent de

la infraestructura que es vulgui utilitzar i, per tant, no se sap el nombre de variables que es necessiten per solucionar el problema. Tot i això, sempre hi ha les mateixes restriccions i en ser un sumatori ha simplificat aquesta dificultat. Un altre dificultat que ha aparegut ha estat el fet que les constants prefixades també depenen de la infraestructura i, per tant, no poden estar definides dins del programa Java. S'han barrejat diferents opcions per solucionar aquest problema, passar totes les constants per paràmetre, però si tenim per exemple 3 dispositius IoT això ja augmenta el nombre de constants a 15 i es complica molt identificar que és cada variable i definir un ordre; l'altre opció que s'ha plantejat i s'ha optat per utilitzar consisteix a elaborar un JSON seguint l'estructura que es mostra al *listing* 3.1. S'ha hagut de fer un pas extra en el solucionador perquè llegeixi i assigni els valors de les constants que es passen en el JSON, al principi per llegir el JSON tenia problemes i no es podia disseccionar les constants; així que finalment s'ha optat per fer una còpia del JSON que es llegeix i analitzar el contingut de la còpia per llegir el valor de les constants.

```
{
    "P": 50,
    "edge": {
        "cpu": 16,
        "texec": 1.2
    },
    "cloud": {
        "cc": 2
    },
    "IoT": {
        "COMPsWorker01": {
            "rp": 1000,
            "cpu": 4,
            "cl": 0.4,
            "texec": 0.3
        },
        "COMPsWorker02": {
            "rp": 5000,
            "cpu": 4,
            "cl": 0.2,
            "texec": 0.8
        }
    }
}
```

Listing 3.1: Exemple JSON entrada solucionador Símplex

3.4.3 Validació

A causa de la falta de temps com s'ha comentat en la secció 3.3.3, s'ha decidit simplificar els tests unitaris i desenvolupar dos tests, un amb un percentatge menor al 50% i un superior al 50%. Amb el percentatge menor es vol comprovar que les tasques es planifiquen correctament en el node remot, mentre que amb el percentatge superior es vol comprovar que la majoria de tasques es planifiquen correctament en el node local.

4 Avaluació del prototip

La part d'avaluació s'ha dividit en dues parts. La primera part consisteix a analitzar cada planificador què afecta el temps de planificar una tasca i com es diferencien les diferents estratègies desenvolupades en comparació amb la política per defecte a COMPSs anomenada *locality*. La segona part consisteix a desplegar una infraestructura amb les tres capes descrites en aquest treball, IoT, edge i Núvol; i llançar una aplicació per estudiar el comportament de la infraestructura i com es comporten els planificadors.

4.1 Anàlisis dels planificadors

Per dur a terme l'anàlisi dels planificadors, primer s'ha estudiat què pot causar una variació en el temps de planificar una tasca. Després de mirar el codi del planificador base, les altres polítiques que disposa COMPSs i discutir-ho amb membres sèniors del grup de desenvolupament, s'ha decidit analitzar el nombre de tasques que es generen, el nombre de paràmetres de les tasques i el nombre de recursos a on es pot planificar una tasca com a causes de la variació en el temps de planificar una tasca. Un cop definits els paràmetres que analitzarem s'ha desenvolupat un programa amb COMPSs que genera un nombre de tasques segons un paràmetre i també, de forma opcional, genera un nombre definit de paràmetres d'entrada per la tasca generada. Amb aquest programa senzill podem controlar dos dels tres paràmetres definits i el nombre de recursos el podem controlar quan es llanci el *Job* al MareNostrum4 mitjançant *Slurm*.

En resum, tenim tres paràmetres que afecten el temps de planificar una tasca i un programa que permet variar aquests paràmetres, toca definir el mètode que seguirem per fer aquesta anàlisi. Primerament, per a cada planificador desenvolupat mirarem com varia el nombre de tasques, farem 4 tests amb 100, 1000, 5000 i 10000 tasques respectivament totes utilitzant 2 recursos i sense paràmetres d'entrada. Per mirar com afecta el nombre de paràmetres d'entrada es duran a terme 3 tests on es generaran 5000 tasques utilitzant 2 recursos i amb 10, 100 i 1000 paràmetres d'entrada cada test. Finalment, per comprovar el nombre de recursos es faran 4 tests amb 5000 tasques, 0 paràmetres i variarem el nombre de recursos entre 4, 8, 16 i 32 recursos en cadascun dels tests. Cal destacar que per cada test es fan múltiples execucions per eliminar l'efecte del *Java-In-Time compiler (JIT)*. Al principi el temps és superior a quan aquest ja està estabilitzat a causa de les optimitzacions que fa del codi en temps d'execució. Per tant, es descarta la primera execució on el JIT no està optimitzat per tal que el temps que aparegui com a resultat sigui més estable. Aquest comportament descrit el podem veure a la figura 4.1. Aquesta figura és

una traça de Paraver on s'està observant el fil del runtime de COMPSs del *Task Dispatcher*. El color turquesa de la gràfica mostra les *Action Update* que es duen a terme durant l'execució de l'aplicació. L'*Action Update* correspon a l'esdeveniment que es produeix quan es planifica una tasca. Un cop explicada la traça, podem observar que en la primera iteració no es pot veure la separació de tasques al color turquesa, mentre que a la segona i tercera iteració es veu clarament com hi ha hagut optimitzacions en temps d'execució. Es pot veure com ha incrementat el nombre de separacions entre les línies turquesa, això indica que el temps que tarda cada tasca a planificar-se ha disminuït.

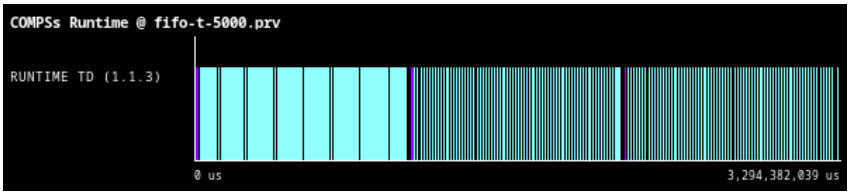


Figura 4.1: Captura de Paraver on s'observa l'efecte del JIT.
Font: Elaboració pròpia

Per treure mètriques utilitzarem l'opció de generar traces que disposa COMPSs utilitzant *Extrae* i a continuació s'analitza la traça amb l'eina *Paraver* per veure i extreure els temps que ha tardat a planificar-se una tasca. Un cop veiem a *Paraver* aquesta mètrica s'exportarà en format *csv*, on tenim per cada instant de temps el nombre de tasques que han tardat aquell temps a planificar-se, per a posterior tractar aquestes dades i generar gràfiques que s'usaran per dur a terme les anàlisis definides prèviament. Per dur a terme l'anàlisi de les dades extretes s'ha desenvolupat un programa amb Python que donat un conjunt de *csv* amb les dades de *Paraver* dissecciona les dades i genera diferents gràfiques. Per exemple, per poder comparar execucions amb diferents nombres de tasques es genera una gràfica amb el valor en format de percentatge tenint en compte el nombre total de tasques generades. Una altra opció consisteix a fer zoom a una part de la gràfica per fixar-nos en la part que ens interessi més. A les gràfiques que es generen, a l'eix x trobem el temps en microsegons i a l'eix y trobem el nombre de tasques o el percentatge de tasques que s'han executat en cada instant de temps de l'eix x. S'utilitza els percentatges quan es comparen diferents nombres de tasques mentre que quan es compara el mateix nombre de tasques generades s'utilitzen els valors absoluts. Cal afegir, a la llegenda de les gràfiques s'indica el planificador que s'utilitza, el nombre de tasques ve indicat pel número després de la lletra **t**. El numero seguit de la lletra **p** indica el nombre de paràmetres. El numero seguit de la lletra **r** indica el nombre de recursos utilitzats. En cas de no trobar la lletra **p** indica que el nombre de paràmetres utilitzat és 0. L'absència de la lletra *r* indica que s'han utilitzat únicament 2 recursos.

Un cop s'hagi analitzat les tres estratègies implementades es farà una comparació dels tres planificadors juntament amb el planificador de *locality* per veure com es diferencien entre ells i

quin és el més ràpid, quin funciona millor amb més recursos, etcètera.

4.1.1 Planificador FIFO

Un cop tenim definida la metodologia és el moment de començar les proves amb el planificador FIFO. Començarem primer analitzant com afecta el nombre de tasques generades, per això hem generat la gràfica que podem veure a la figura 4.2. En aquesta gràfica podem veure que la gran majoria de tasques es planifiquen de forma pràcticament instantània. A la figura 4.3, que és una ampliació de la figura 4.2, podem veure que al cap d'aproximadament 100 microsegons hi ha un segon pic de tasques que es planifiquen i els pics màxims són dels tests amb més tasques. A més, a la figura 4.4, que mostra un graf del percentatge de tasques executades en l'instant 0 de temps. Podem veure que com menys tasques s'executen més elevat és el percentatge de tasques que tarden pràcticament 0 a planificar-se. En conclusió, en l'estratègia FIFO augmentar el nombre de tasques fa que el temps de planificar les tasques també augmenti.

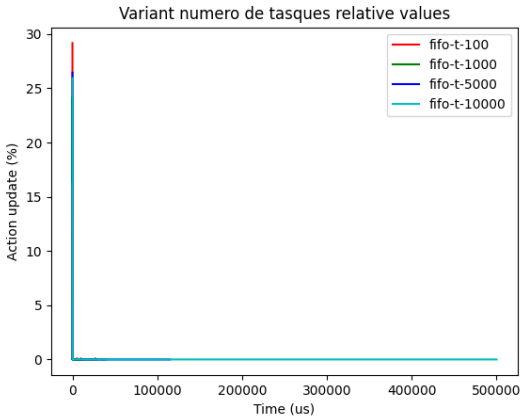


Figura 4.2: Planificador FIFO variant el nombre de tasques.
Font: Elaboració pròpia

El segon paràmetre que analitzarem la manera a què afecte aquest planificador és el nombre de paràmetres d'entrada de cada tasca. Com s'ha explicat anteriorment s'ha executat 5000 tasques amb diferents paràmetres d'entrada i comparem els resultats obtinguts. A la figura 4.5 podem observar la gràfica generada variant el nombre de paràmetres, però no ens aporta massa informació a causa de la gran quantitat d'instants de temps capturats, com veiem la majoria de les tasques es planifiquen amb molt poc temps en la línia vertical a l'instant 0. Per observar millor amb més detall com afecta s'ha fet zoom a aquesta part i s'ha obtingut com a resultat la figura 4.6 on podem observar que hi ha diferents pics i com més a la dreta és el pic més paràmetres d'entrada hi ha. En resum, podem afirmar que el nombre de paràmetres alenteix la planificació de tasques i, per tant, és més costós de planificar si hi ha paràmetres d'entrada.

Finalment, s'ha estudiat com afecta el nombre de recursos sobre els quals es poden planificar

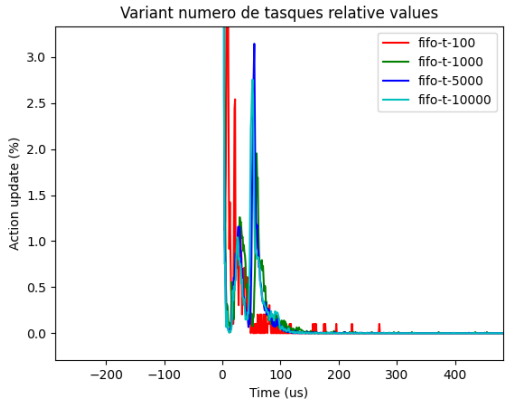


Figura 4.3: FIFO variant nombre de tasques amb zoom a la part inferior esquerra.
Font: Elaboració pròpia

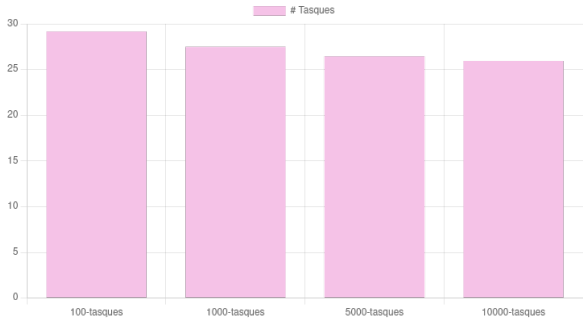


Figura 4.4: Gràfic amb el percentatge de tasques a l'instant 0.
Font: Elaboració pròpia

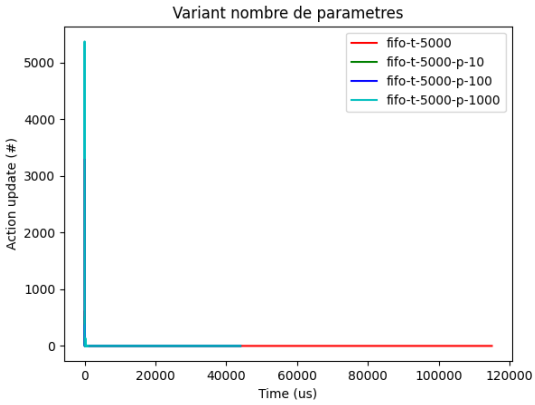


Figura 4.5: FIFO variant nombre de paràmetres.
Font: Elaboració pròpia

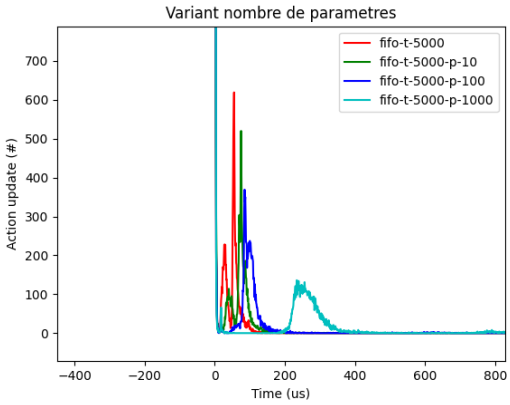


Figura 4.6: FIFO variant nombre de paràmetres amb zoom a la part inferior esquerra.
Font: Elaboració pròpia

tasques. Aquesta és una avaluació que es fa únicament per aquesta estratègia, ja que les altres no estan dissenyades per executar-se en múltiples recursos a part del local i un remot. Centrem-nos en aquesta política. A la figura 4.7 tenim la gràfica generada en executar el test explicat en la secció anterior. Tot i que per veure resultats s'ha d'observar la figura 4.8 on podem veure la figura de l'esquerra, però fent zoom a la part inferior esquerra i es pot percebre com no hi ha massa diferència amb el nombre de recursos. Per tant, podem dir que el nombre de recursos quasi afecta aquesta estratègia.

Per finalitzar, aquest planificador és molt ràpid a planificar les tasques es pot observar que pràcticament hi ha sempre dos pics el primer quasi al 0 i un segon proper a aquest depenent del nombre de tasques, paràmetres o recursos que s'utilitzin, però amb aquests dos pics es planifica

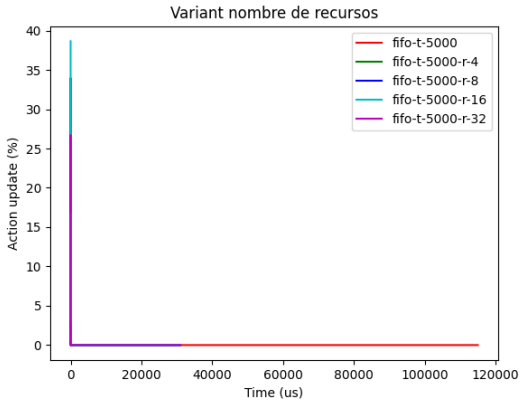


Figura 4.7: FIFO variant nombre de recursos.
Font: Elaboració pròpia

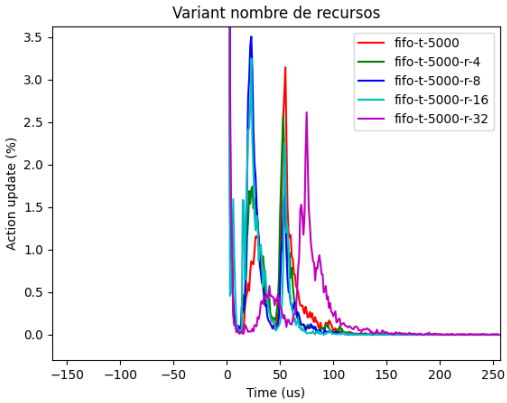


Figura 4.8: FIFO variant nombre de recursos amb zoom a la part inferior esquerra.
Font: Elaboració pròpia

la gran majoria de les tasques. Per tant, l'objectiu de desenvolupar un planificador ràpid que no tingui massa en compte el nombre de recursos ha estat satisfet.

4.1.2 Planificador backpressure

A continuació, s'ha analitzat la segona estratègia desenvolupada en aquest projecte, que consisteix a fer servir backpressure per planificar tasques. S'han fet servir els mateixos tests que amb el FIFO.

Començarem analitzant com afecta el nombre de tasques. A la figura 4.9 s'observa que la gran majoria de tasques tarden a planificar-se prop de l'instant 0. Tot i que si fem zoom a la part propera al zero com es veu a la figura 4.10 podem observar que com més elevat és el nombre de tasques més temps tarda a planificar-les. Això és degut al fet que per cada tasca que ha de planificar analitza totes les restants per mirar quina és l'òptima per planificar. A diferència del FIFO, no trobem pics sinó línies que van disminuint amb el temps, ja que quan una tasca es planifica aquesta surt del sistema i en la pròxima iteració ja no es tindrà en compte.

Pel que fa a l'impacte del nombre de paràmetres en aquest planificador, la figura 4.11 mostra la gràfica obtinguda. Però un cop més ens hem de centrar en la part inferior esquerra i analitzar com s'ha comportat el planificador en aquest test. A la figura 4.12 es troba la part inferior esquerra i es pot observar com el nombre de paràmetres no afecta massa a l'hora de planificar les tasques, ja que totes les opcions que s'han executat han obtingut resultats similars i no es veu clarament com tenir 1000 o 0 paràmetres afecta considerablement la planificació de tasques. L'efecte del nombre de tasques és més gran que el del nombre de paràmetres.

En últim lloc, analitzarem com el nombre de recursos pot afectar a aquesta estratègia. En

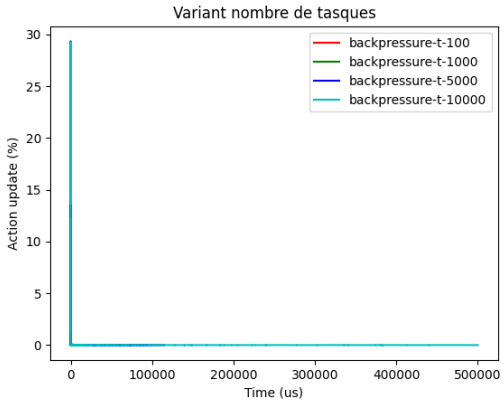


Figura 4.9: Backpressure variant nombre de tasques.
Font: Elaboració pròpia

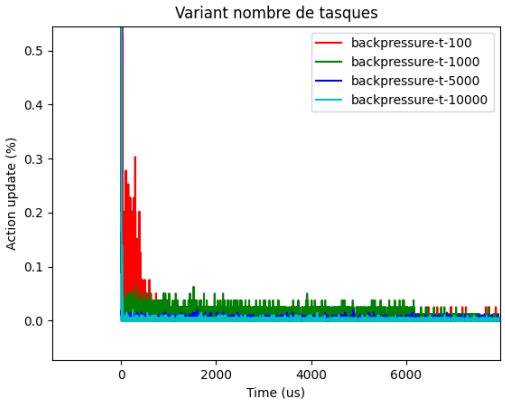


Figura 4.10: Backpressure variant nombre de recursos amb zoom a la part inferior esquerra.
Font: Elaboració pròpia

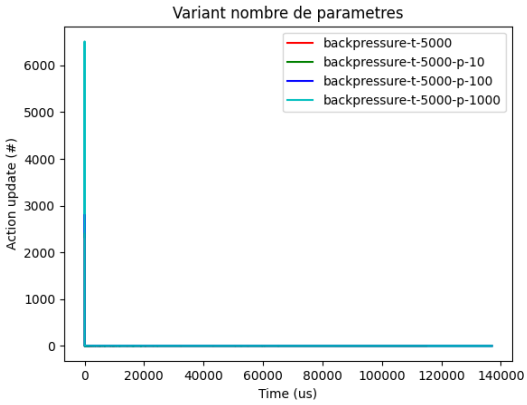


Figura 4.11: Backpressure variant nombre de paràmetres.
Font: Elaboració pròpia

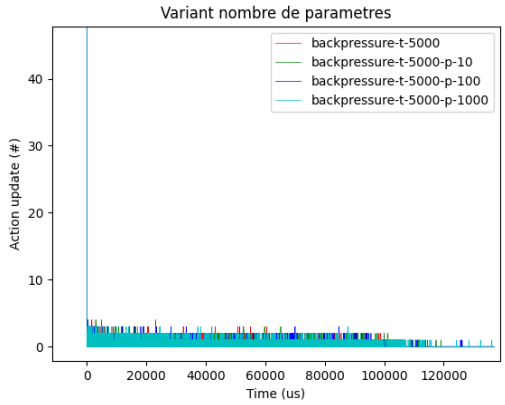


Figura 4.12: Backpressure variant nombre de paràmetres amb zoom a la part inferior esquerra.
Font: Elaboració pròpia

aquest test falta mirar com es comporta amb 32 recursos per culpa de què el *Job* no ha entrat a MareNostrum 4 i no hi ha temps per esperar que aquest finalitzi. Tot i això, podem veure com es comporta amb 2, 4, 8 i 16 recursos. A la figura 4.13 podem veure la gràfica obtinguda tot i que no es veu res que cridi l'atenció. Per estudiar millor el comportament, a la figura 4.14 podem veure una ampliació de la figura 4.13 on es veu com amb el temps hi ha tasques de les execucions amb més recursos. Per tant, el nombre de recursos afecta quan s'ha de planificar una tasca, però no té el mateix impacte que el nombre de tasques.

En conclusió, aquest planificador és dependent del nombre de tasques que tingui el temps a planificar una tasca pot ser molt elevat i aquest fet que tardi massa a planificar les tasques

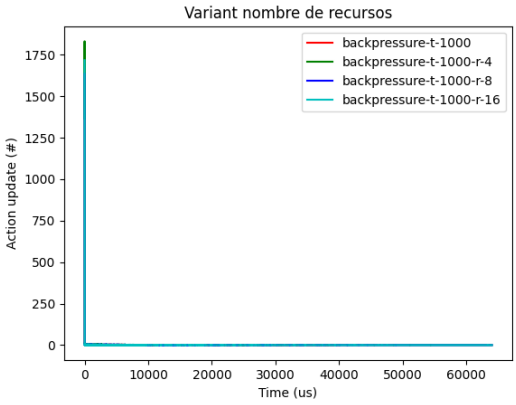


Figura 4.13: Backpressure variant nombre de recursos.
Font: Elaboració pròpia

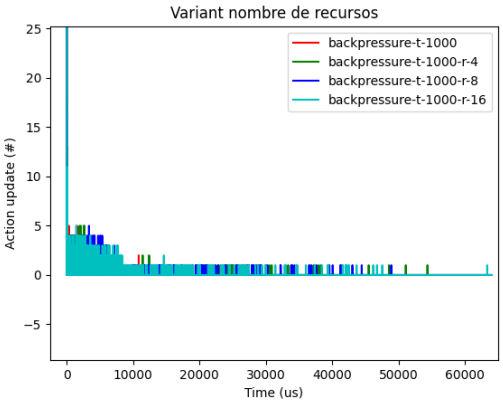


Figura 4.14: Backpressure variant nombre de recursos amb zoom a la part inferior esquerra.
Font: Elaboració pròpia

pot comportar que no es puguin omplir a temps els recursos. Per exemple, si tenim dos nodes de MN4, és a dir, 96 cores i les tasques tarden 15 segons a executar-se, podem calcular el temps màxim que hauria de tardar el planificador a planificar cada tasca hauria de ser $15/96 = 156\ ms$. Per tant, si el planificador tarda més de 156 ms en planificar una tasca no s'omplirà la infraestructura i es perdrà rendiment, ja que sempre hi haurà algun recurs que no es podrà omplir perquè un recurs que s'estava utilitzant per executar una tasca haurà acabat abans que tots els altres es puguin omplir. Aquest problema s'ha de tenir en compte en infraestructures grans, ja que com més recursos disposi la infraestructura, més ràpid s'hauran de planificar les tasques.

4.1.3 Planificador teoria de jocs

En aquest punt, s'analitza la tercera i última estratègia implementada, la basada en teoria de jocs. Per aquesta igual que amb la de backpressure, només ens centrem en el nombre de paràmetres i el nombre de tasques. Ja que com s'ha explicat en l'apartat corresponent (vegeu 3.4), només disposem del node IoT i el servidor edge per enviar tasques, és per això que no té sentit analitzar com es comporta amb un nombre elevat de recursos. S'ha començat analitzant el nombre tasques que es generen quin efecte tenen en aquesta política. S'ha seguit el mateix test que s'ha descrit al principi d'aquest capítol i s'ha obtingut la gràfica de la figura 4.15. Tot i això, per veure el comportament del planificador s'ha d'observar la part superior i inferior esquerra per veure realment quan s'executen les tasques i la quantitat. A la figura 4.17 podem veure un graf de barres que indica el percentatge de tasques que la seva planificació ha estat quasi instantània. S'aprecia que com més tasques hi ha, el percentatge és menor al voltant de l'instant de temps 0. A més, a la figura 4.16 es veu com igual que al FIFO hi ha diferents pics i s'observa que els pics més allunyats de l'instant 0, és a dir, les tasques que tarden més

planificar corresponen al test amb més tasques generades. Per tant, podem veure com el nombre de tasques afecta directament al temps a planificar les tasques. Com més tasques més temps es tarda.

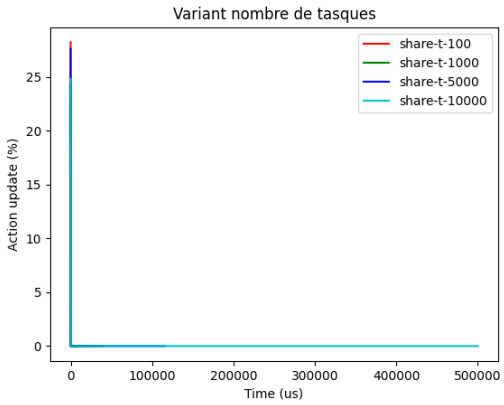


Figura 4.15: Teoria de jocs variant nombre de tasques.
Font: Elaboració pròpia

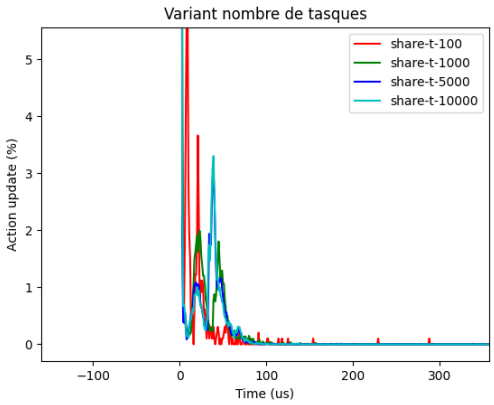


Figura 4.16: Teoria de jocs variant nombre de tasques amb zoom a la part inferior esquerra.
Font: Elaboració pròpia

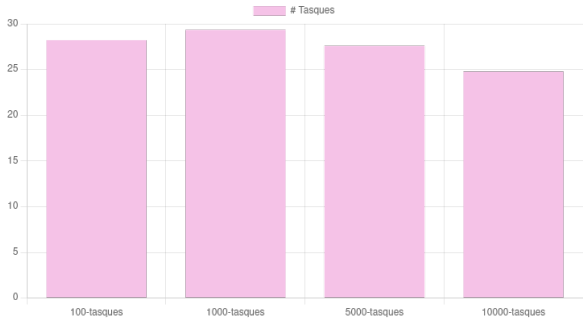


Figura 4.17: Gràfic Teoria de jocs que indica el percentatge de tasques en l'instant 0.
Font: Elaboració pròpia

Seguint amb el planificador de teoria de jocs, toca analitzar com es comporta variant el nombre de paràmetres d'entrada de les tasques. Podem veure a la figura 4.18 la gràfica generada, però per extreure conclusions s'ha d'analitzar més en detall la part inicial d'aquesta, ja que és el punt on es planifica el nombre més gran de tasques. Concretament, a la figura 4.19 trobem zoom a la part inferior esquerra i trobem exactament el mateix comportament que ens hem trobat en el FIFO i és completament diferent del backpressure. Podem veure un altre cop aquests pics característics que indiquen que hi ha dues tongades de planificació de tasques i aquesta segona tongada es veu clarament com més paràmetres d'entrada tinguin les tasques més a la dreta es troben i, per tant, més tarden a planificar-se.

En resum, el planificador de teoria de jocs es comporta de forma molt similar a l'estratègia FIFO,

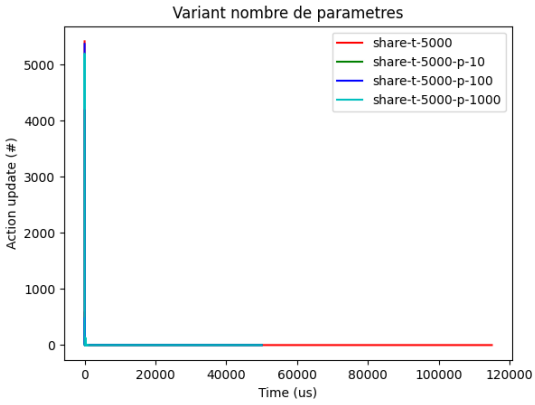


Figura 4.18: Teoria de jocs variant nombre de paràmetres.
Font: Elaboració pròpia

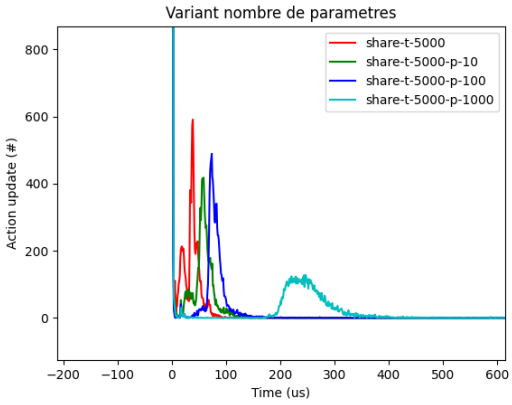


Figura 4.19: Teoria de jocs variant nombre de paràmetres amb zoom a la part inferior esquerra.
Font: Elaboració pròpia

tot i això, l'ús de cada planificador és en àmbits diferents. Tanmateix, veiem com s'ha generat un planificador que no requereix massa còmput, ja que les tasques es planifiquen de pressa tot i afegir tasques i paràmetres, ja és el que es buscava en un planificador per la capa de IoT que sigui capaç de planificar tasques ràpides i que no requereixi massa cost computacional perquè molts dispositius IoT manquen d'aquesta característica.

4.1.4 Comparació dels planificadors

Per acabar la comparació entre els planificadors, generarem una gràfica que es pot veure a la figura 4.20 on hem agafat els tres planificadors desenvolupats, a més del que s'utilitza per defecte a COMPSs per veure com es diferencien entre ells. S'ha decidit utilitzar amb 5000 tasques perquè com s'ha vist en les avaluacions anteriors, amb 5000 tasques ja es pot veure com aquest nombre de tasques té un efecte al temps de planificar-les. Tot i que per veure més en detall com es comporta cada planificador ens hem de centrar en dues zones de la gràfica. La primera zona és la que es pot veure a la figura 4.21 que correspon a la part superior esquerra. En aquesta figura s'observa les estratègies de FIFO i teoria de jocs són les més ràpides en planificar tasques, ja que estan més properes a l'instant 0, mentre que backpressure i *locality* tenen un pic de tasques al cap d'uns microsegons. Això ja ens indica que d'entrada les dues primeres estratègies són més ràpides, a més a més, si mirem a la figura 4.23, que es tracta d'un zoom amb precisió de desenes a l'eix y de la part inferior esquerra, es pot veure com predominen les tasques del backpressure i *locality* es continuen planificant en instants elevats de temps. En contra, no s'observa el mateix comportament del FIFO i teoria de jocs, aquests tenen un pic abans de l'instant 100 com es pot veure a la figura 4.22, que aquesta se centra en centenars d'unitats en ambdós eixos, i entre aquests dos es veu clarament que el de teoria de jocs és el més ràpid.

En conclusió, el planificador de teoria de jocs és el més ràpid, ja que la gran majoria de tasques es planifiquen en un dels dos pics vists, un a l'instant 0 i l'altre al voltant del 50. Seguidament, el segueix el planificador FIFO que també té dues puntes on es planifiquen tasques, una al 0 i l'altre més tard que la de l'estratègia de teoria de jocs. Per acabar, trobem el de backpressure que és una mica més ràpid que el de *locality* ja que el primer pic de tasques proper a l'instant 2 de temps és més elevat que el de la política de *locality*. Tot i això, aquests dos planificadors són bastant lents perquè no tenen un nombre de pics sinó que els temps són més variables i es poden allargar fins a tardar segons en funció del nombre de tasques i característiques del recurs com s'ha vist prèviament.

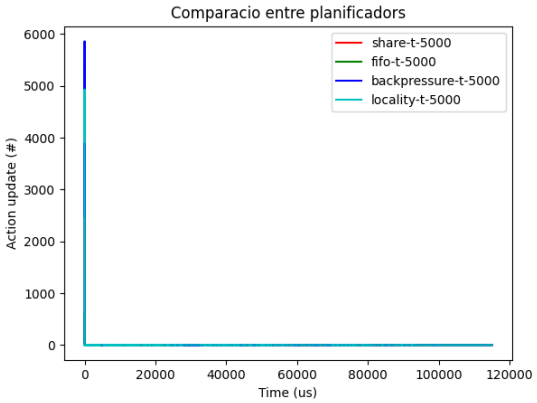


Figura 4.20: Comparació dels diferents planificadors implementats i el per defecte.
Font: Elaboració pròpia

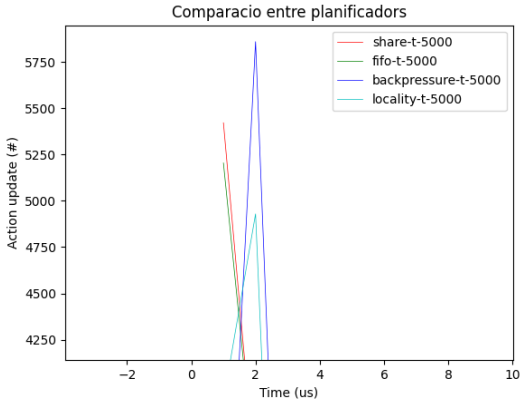


Figura 4.21: Comparació dels diferents planificadors implementats i el per defecte amb zoom a la part superior esquerra.
Font: Elaboració pròpia

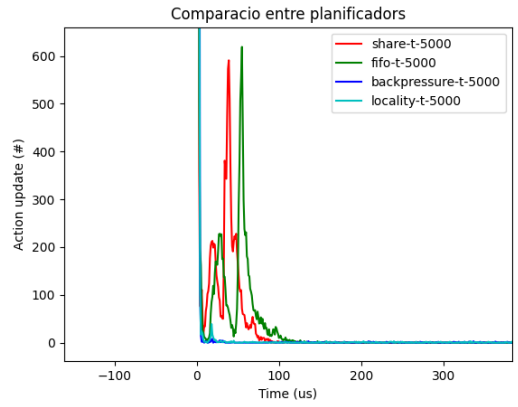


Figura 4.22: Comparació dels diferents planificadors implementats i el per defecte amb zoom a la part inferior esquerra.
Font: Elaboració pròpia

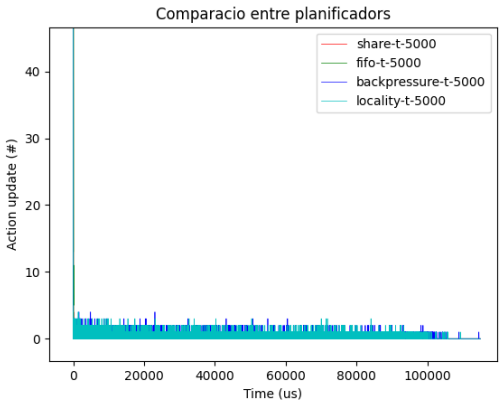


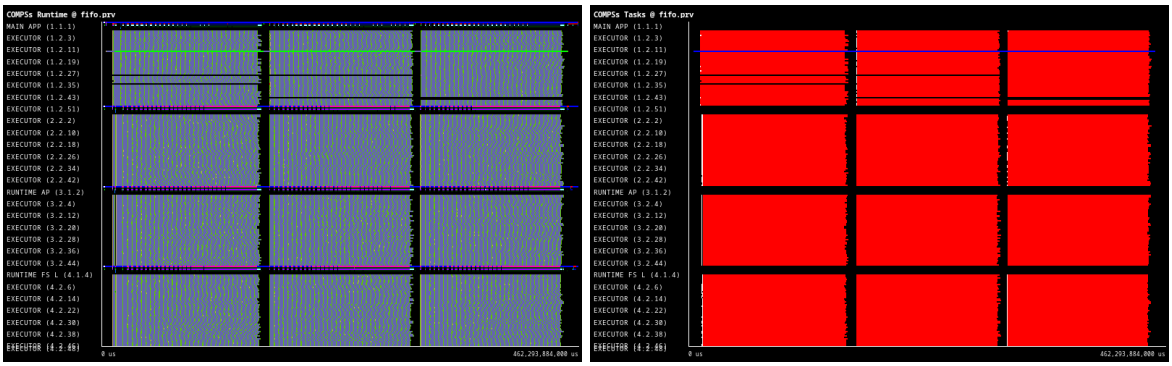
Figura 4.23: Comparació dels diferents planificadors implementats i el per defecte amb més zoom a la part inferior esquerra.
Font: Elaboració pròpia

4.2 Anàlisis en infraestructura Cloud

Per fer aquest test, s'ha utilitzat l'algorisme de classificació *Random Forest*[26]. S'ha executat al MareNostrum 4 utilitzant 6144 estimadors, 4 nodes (192 cores) i utilitzant els planificadors de locality i FIFO. Aquest test es fa amb l'objectiu d'observar diferències de rendiment entre els dos planificadors i analitzar-les. Per aquest test es volia un programa que generes moltes tasques per poder veure com es comporta el planificador. Aquest tipus de programes que generen moltes tasques tendeixen a executar-se al Núvol i en superordinadors, és per això que s'ha utilitzat l'estratègia FIFO, ja que està dissenyada per aquest nivell. S'ha utilitzat un *Random Forest* que és un entrenament d'un model de IA, necessita moltes dades centralitzades i gran capacitat de càlcul. Per això es tira al Núvol. També dins del grup de recerca s'està treballant amb un programa que utilitza aquest algorisme i, per tant, no s'ha de gastar temps en desenvolupar-lo. A més, s'havien observat problemes de rendiment amb el planificador per defecte.

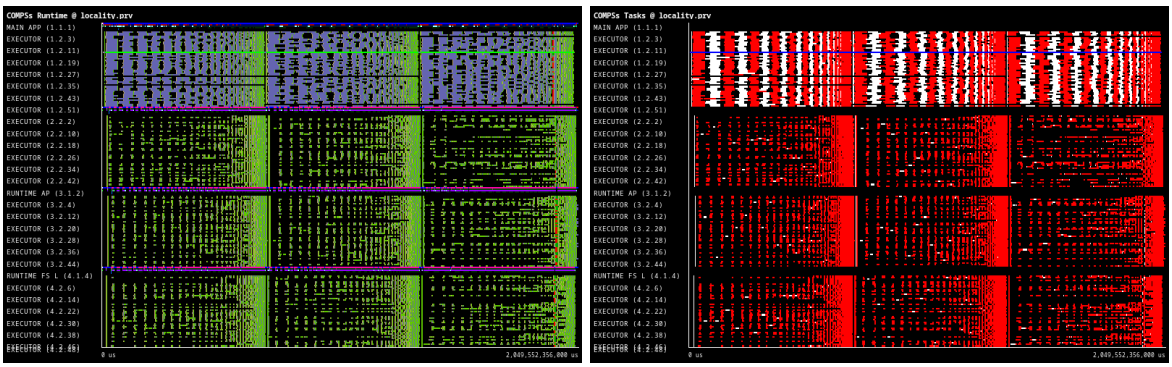
Per fer l'anàlisi s'ha obtingut una traça per cada planificador. Per cada traça es vol observar el temps que es tarda a planificar una tasca i l'ocupació que es fa dels recursos, és a dir, si es poden planificar les tasques prou ràpides per ocupar totes les CPUs. A primer vista si observem les figures 4.24 i 4.25 podem veure que la corresponent al planificador locality té més parts negres. Això vol dir que hi ha moments on no està fent res aquell executor. Si ens fixem en la configuració de tasques (figures 4.24b i 4.25b) podem veure que la primera és pràcticament de color vermell, mentre que l'altre té bastants parts negres. Això ens indica que el planificador FIFO és més ràpid a l'hora de planificar tasques i quan una tasca acaba no s'ha d'esperar gaire temps a què el planificador planifiqui la pròxima tasca. El mateix es pot observar a les figures amb la configuració de runtime (vegeu figures 4.24a i 4.25a) on es pot veure que en el cas de la política de locality la part de dalt, és a dir, el recurs local és on podem veure que hi ha més tasques executant-se, ja que el color lila indica que una tasca està corrent. En els altres recursos no es veu tanta quantitat de tasques, ja que el locality està saturat de tasques i no és capaç de planificar-les prou ràpid per omplir tots els recursos. En canvi, si mirem la traça del FIFO (figura 4.24a) podem veure com el color lila és pràcticament visible a tots els recursos i no hi ha espais en negre, tret de quan acaba una iteració.

A més, si mirem les traces de les figures 4.26 i 4.27 podem veure els esdeveniments que s'han produït a l'agent 1. Si ens fixem en el fil 1.1.3, que mostra els esdeveniments del planificador, s'observa com la traça corresponent al locality no té cap forat negre indicant que el planificador està tota l'estona funcionant sense parar, mentre que a la traça corresponent al planificador FIFO es veu com hi ha moments que no ha de planificar tasques perquè té tots els recursos ocupats. Això ens indica que hi ha un problema de planificadors que afecta el temps d'execució del programa. Aquest problema es pot solucionar utilitzant la política FIFO desenvolupada en aquest treball.



(a) Configuració runtime (b) Configuració tasques

Figura 4.24: Traça de Paraver amb el planificador FIFO.
Font: Elaboració pròpia



(a) Configuració runtime (b) Configuració tasques

Figura 4.25: Traça de Paraver amb el planificador locality.
Font: Elaboració pròpia

Un cop analitzades les traces i veient el comportament de cada planificador. Per acabar la comparació ens fixarem en els temps que ha tardat cada planificador a executar el programa. A la figura 4.28 tenim a l'eix vertical el temps d'execució (ms) i a l'eix horitzontal cadascuna de les tres iteracions que s'han fet en l'execució del programa. S'observa com el FIFO és molt més ràpid que l'estratègia de locality a l'hora de planificar les tasques i, per tant, en l'execució total del programa. A més, és capaç d'omplir de forma més eficient els recursos què disposa. El temps mitjà d'execució amb el planificador FIFO és de 142 segons, mentre que amb locality és de 660 segons. En conclusió el planificador FIFO és quatre cops més ràpid que el locality en aquesta aplicació en entorns Cloud.

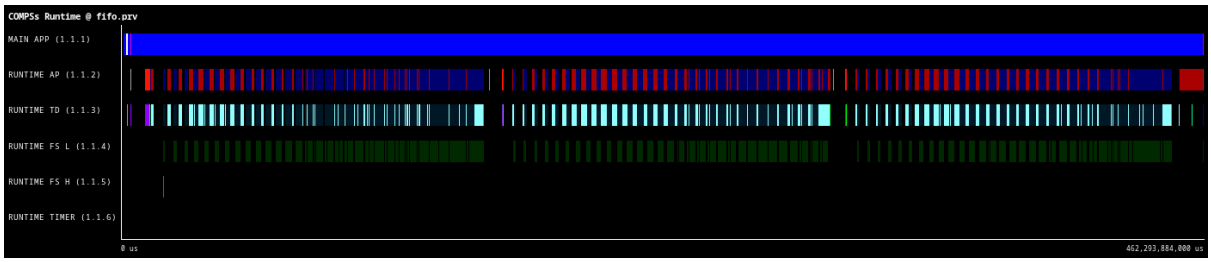


Figura 4.26: Configuració runtime de l'agent 1 FIFO.
Font: Elaboració pròpia

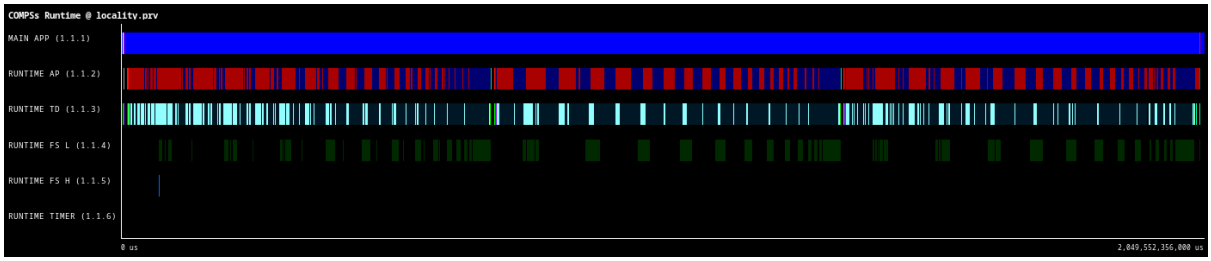


Figura 4.27: Configuració runtime de l'agent 1 locality.
Font: Elaboració pròpia

Figura 4.28: Comparació de temps FIFO i locality.
Font: Elaboració pròpia

4.3 Infraestructura IoT-edge-Cloud

Per fer l'avaluació desplegant una infraestructura amb les 3 capes, primer s'ha de definir la infraestructura que s'utilitzarà. En l'àmbit d'aquest treball, reduirem la infraestructura a un únic servidor edge connectat al Núvol. En casos reals poden haver-hi desenes o fins i tot centenars de servidors edge, i d'aquest servidor edge hi pengen tres dispositius IoT com podem veure a la figura 4.29. Aquests dispositius IoT poden ser molt diversos, des de sensors que capturen informació a un mòbil o una Raspberry Pi. Per fer el desplegament de la infraestructura descrita s'han definit

els dispositius que s'utilitzaran per a cada capa. Per a la capa IoT, idealment haurien de ser tres dispositius diferents per simular diferents arquitectures i generació de tasques. Tot i que per simplificar el test hem decidit agafar el mateix per als tres dispositius, concretament utilitzarem tres *Raspberry Pi 3 Model B+* com la de la figura 4.30. En aquesta capa s'utilitzarà l'estratègia de Teoria de jocs per planificar les tasques. Per la cada edge ens hem decantat per utilitzar una *Nvidia Jetson TX1* com la de la figura 4.31, ja que és més potent que una Raspberry Pi i, per tant, compleix el requisit que el servidor edge ha de disposar de més capacitat de càlcul que els dispositius IoT. A més, s'utilitzarà el planificador de Backpressure en aquesta capa. Finalment, per la capa superior després d'analitzar diferents alternatives, hem optat per utilitzar el servei d'Azure com a Núvol juntament amb la política FIFO per planificar les tasques.

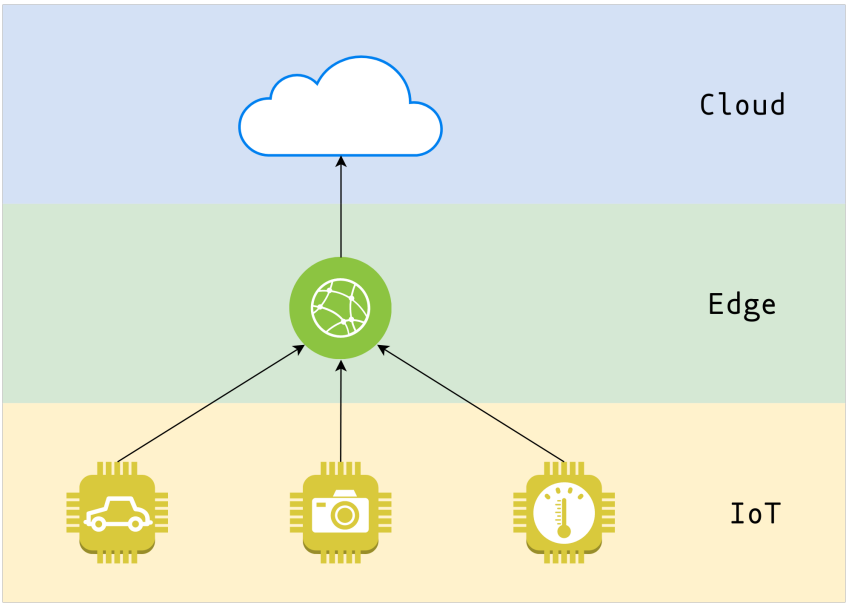


Figura 4.29: Infraestructura utilitzada per a l'avaluació.
Font: Elaboració pròpia

El següent pas que s'ha de realitzar un cop definida la infraestructura és preparar els dispositius que s'utilitzaran instal·lant el software necessari per a fer les proves. En el cas de les raspberry s'ha fet una instal·lació de COMPSs en una de les 3 disponibles i un cop completada amb èxit s'ha creat una imatge base per posar a les altres. Tot i això, hi ha hagut diverses complicacions durant el procés. Per exemple, no hi havia una versió de Java compatible als repositoris de raspberry i s'ha hagut d'utilitzar el JDK d'Oracle a diferència del que s'utilitzava fins ara que era d'OpenJDK. Això ha provocat errors a l'hora de comprovar el resultat de la instal·lació i s'ha hagut d'afegir un *flag* a Java. En aquest cas s'ha utilitzat la variable d'entorn *JAVA_TOOL_OPTIONS* quan aquest s'executa per tal d'activar una opció que per defecte ja s'utilitzava amb l'altre JDK. Una possible dificultat que ens podíem haver trobat era el fet que la raspberry utilitza arquitectura ARM mentre que amb els entorns que s'han treballat fins al

moment eren x86, tanmateix, això no ha suposat cap problema durant la instal·lació.

El següent dispositiu que s'ha de configurar és la Jetson que també utilitza Linux com a sistema operatiu i té una arquitectura ARM. Contràriament a les raspberry, sí que disposava de la versió habitual de Java amb el JDK d'OpenJDK. Tot i això, han aparegut altres problemes, com per exemple durant la fase d'execució Java no tenia suficient memòria assignada i, per tant, fallava l'execució. Això s'ha pogut arreglar assignant-li més memòria a Java mitjançant la variable d'entorn *_JAVA_OPTIONS*.

A continuació, es fa una prova senzilla per comprovar si es pot establir connexió entre els dispositius IoT i el servidor edge. Es crea una topologia utilitzant una raspberry que genera tasques i se li envien a la Jetson i es comprova que aquesta les rebi i s'executin correctament. Només s'utilitza una raspberry, ja que en tenir totes la mateixa imatge, si funciona una la resta també ho faran. El següent pas consisteix a preparar l'entorn al Núvol, en concret crearem una màquina virtual a Azure i allà hi instal·larem la mateixa versió de COMPSs que s'ha posat a les Raspberry Pi i la Jetson. Aquesta màquina virtual s'ha fet utilitzant una imatge d'Ubuntu 20.04 i amb una màquina de 2 CPUs i 8 GB de memòria RAM més que suficient per al test que es vol fer. L'únic problema/dificultat que ens hem trobat durant el procés de creació i instal·lació de la màquina virtual ha estat familiaritzar-se amb la interfície d'Azure i posar les normes de xarxa concretes per poder tenir accés als ports que utilitzen els agents per a l'intercanvi d'informació.

Figura 4.30: Raspberry Pi 3 Model B+.
Font: https://www.raspberrypi.org/

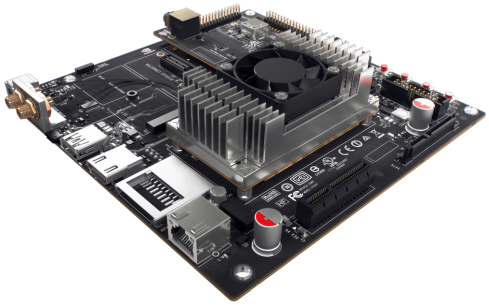


Figura 4.31: Nvidia Jetson TX1 Development Kit.
Font: https://developer.nvidia.com/embedded/jetson-tx1-developer-kit

Un cop preparats els diferents dispositius amb el software necessari, s'han desplegat els agents i s'ha comprovat que estigui ben fet el desplegament i no hi hagi problemes de permisos relacionats amb internet. El desplegament s'ha fet en una xarxa local i Azure. A la xarxa local tenim les raspberry Pi i la Jetson, i d'aquests només la Jetson ha de tenir accés a Internet per poder enviar tasques, si fa falta, al Núvol. A causa d'això, s'ha hagut de configurar el router de la xarxa local perquè la Jetson sigui accessible per a Azure. Després d'aquesta modificació ja tenim la infraestructura funcional, el següent pas consisteix a definir un cas d'ús de laboratori

sintètic per aquesta infraestructura. Cal remarcar, aquest cas d'ús no es desenvoluparà, ja que podria ser tranquil·lament un TFG causa de la càrrega de feina. En l'àmbit d'aquest projecte es presentarà un esquema que s'ajusta a diversos casos d'ús reals que es podrien beneficiar de la infraestructura. Alguns exemples d'aquests casos d'ús podria ser càmeres de trànsit en una ciutat, per a seguretat en aeroports, identificar la població d'ocells al bosc. Pel que fa a identificar la població d'ocells, aquest exemple consisteix a desplegar en l'àmbit de IoT dispositius per tot el bosc que siguin capaços de detectar i registrar els cants dels ocells. A posterior, identificar quin tipus d'ocell es tracta l'enregistrament fet i un cop recol·lectats els ocells identificats podries saber la població d'ocelles en un bosc amb les dades que ha recollit cada dispositiu.

El punt següent és el tipus de tests que s'utilitzaran per comprovar el funcionament dels planificadors en la infraestructura IoT-edge-Cloud. S'han dissenyat tres tipus de tests, el primer consisteix a utilitzar únicament la capa d'edge per planificar les tasques i que s'executin únicament en aquesta tasca. El segon test consisteix a utilitzar les dues capes més baixes, carregar l'edge sense saturar-lo i que les tasques restants s'executin en local. Per acabar, el tercer tes consisteix a utilitzar les tres capes seguint la idea de no saturar la cada de l'edge i que es pugui complir el període establert.

Pel que fa al primer test, s'ha utilitzat la configuració que es mostra a la taula 4.1 amb l'objectiu de generar suficients tasques perquè s'executin únicament al servidor edge i aquest no es col·lapsi. Amb aquest test esperem veure com a resultat que la gran majoria de tasques han estat planificades i executades a l'edge. A la configuració s'ha assumit que el temps del servidor edge és 10 cops més ràpid que els dispositius IoT, ja que disposa d'una targeta gràfica que acostuma a ser 10 cops més ràpida que una CPU. Un cop dissenyada la configuració inicial, s'ha utilitzat el solucionador algebraic per mirar quins són els percentatges per obtenir el mínim cost prioritzant l'execució a l'edge. El solucionador reporta que el percentatge ha de ser 100% al servidor edge com esperàvem. S'ha deixat el programa de proves executant durant 20 minuts i s'ha obtingut com a resultat la gràfica de la figura 4.32. Com podem veure la gran majoria de tasques s'han executat al servidor edge que ja és el que s'esperava.

Període				3600
Dispositiu	rp	CPU	cost	tExec
IoT-1	3600	4	0.4	1
IoT-2	3600	4	0.4	1
IoT-3	3600	4	0.2	1
edge	-	4	0	0.2
Cloud	-	-	2	0.4

Taula 4.1: Configuració exemple edge

Quant al segon test, s'ha buscat afegir una capa més en comparació amb el primer, la capa IoT. En aquesta configuració s'ha buscat un nombre de tasques generades que no sobrecarregui

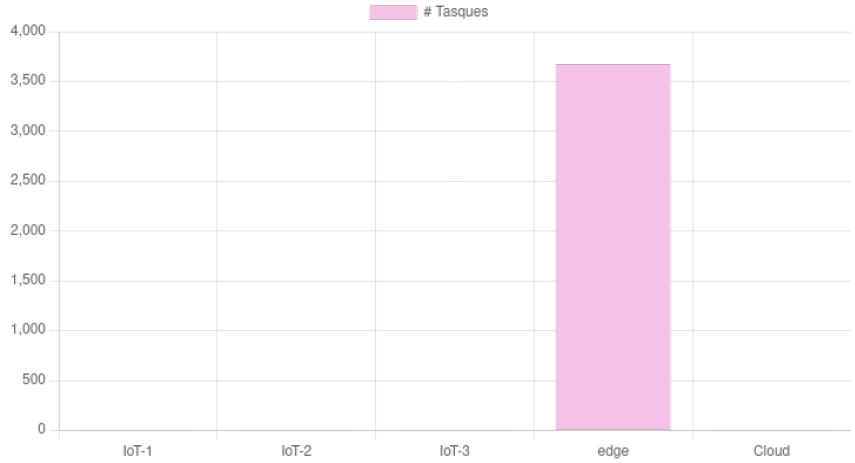


Figura 4.32: Gràfica amb el nombre de tasques executades amb la configuració edge.
Font: Elaboració pròpia

l'edge i, per tant, no enviï tasques al Núvol. La configuració usada es pot trobar a la taula 4.2. Respecte a la configuració del primer test s'ha reduït el període P i ha augmentat el nombre de tasques que genera cada dispositiu IoT. S'ha evitat generar un nombre elevat de tasques perquè no hi hagi tasques que s'enviïn al Núvol. El solucionador et dona el percentatge de tasques que haurien d'executar-se en local (λ), el percentatge de tasques que haurien d'executar-se a l'edge (ϵ) i el percentatge que va al Núvol (κ). El resultat del solucionador algebraic en aquesta configuració ha estat el següent:

- **IoT-1.** Local(λ) = $1.\bar{6}\%$, Edge(ϵ) = $98.\bar{3}\%$ i Núvol(κ) = 0%
- **IoT-2.** Local(λ) = $26.\bar{6}\%$, Edge(ϵ) = $73.\bar{3}\%$ i Núvol(κ) = 0%
- **IoT-3.** Local(λ) = 30%, Edge(ϵ) = 70% i Núvol(κ) = 0%

Per tant, de forma teòrica no s'hauria d'executar cap tasca al Núvol, però, tot i això, com podem veure a la figura 4.33 hi ha tasques que han estat executades al Núvol. Això és degut al fet que el servidor edge té la seva pròpia política de planificació (Backpresssure) i, per tant, decideix com planifica ell les tasques. Malgrat tot, les tasques que s'han d'executar en local i les que envia al servidor edge són els resultats esperats. En conclusió, en aquest test la capa IoT reparteix les tasques com s'espera que ho faci amb els resultats obtinguts del solucionador algebraic.

Pel que fa a l'últim test, s'ha volgut utilitzar la infraestructura sencera utilitzant les tres capes descrites, IoT, edge i Núvol. Per això, s'ha generat una configuració que generi suficient tasques per enviar al Núvol, però prioritzant sempre el servidor edge, per fer això s'ha augmentat el temps d'execució de la capa IoT. A més, com es pot veure a la taula 4.3 la configuració utilitzada, hi ha un dispositiu IoT amb el cost inferior als altres per prioritzar que aquest dispositiu executi un major nombre de tasques en local. En casos reals, això es podria donar perquè s'utilitza

Període				720
Dispositiu	rp	CPU	cost	tExec
IoT-1	7200	4	0.4	2
IoT-2	5400	4	0.4	2
IoT-3	4800	4	0.4	2
edge	-	4	0	0.2
Cloud	-	-	2	0.4

Taula 4.2: Configuració exemple IoT - edge

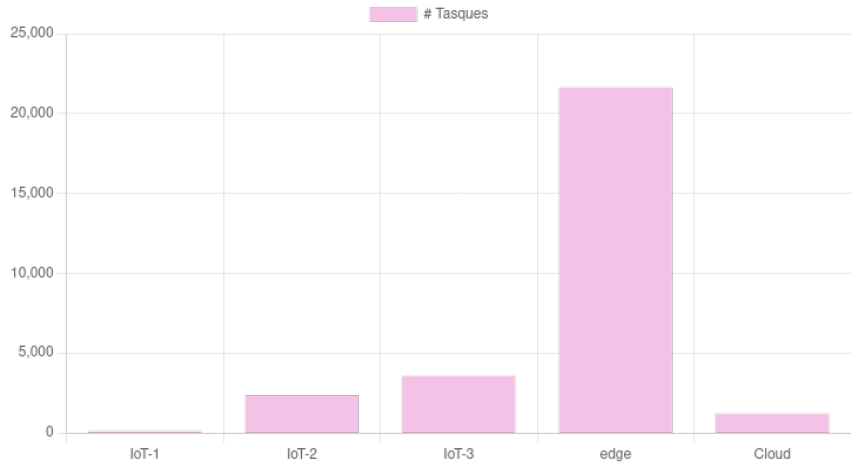


Figura 4.33: Nombre de tasques executades a cada dispositiu amb la configuració IoT-edge. Font: Elaboració pròpia

una nova versió de maquinari que redueix el consum energètic per exemple. Passada aquesta configuració pel solucionador algebraic s'han obtingut els següents percentatges:

- **IoT-1.** Local(λ) = 10%, Edge(ϵ) = 78.$\bar{3}$% i Núvol(κ) = 11.$\bar{6}$%
- **IoT-2.** Local(λ) = 13.$\bar{3}$%, Edge(ϵ) = 86.$\bar{6}$% i Núvol(κ) = 0%
- **IoT-3.** Local(λ) = 15%, Edge(ϵ) = 85% i Núvol(κ) = 0%

Veient els resultats obtinguts a la figura 4.34 podem veure com s'ha respectat bastant els percentatges i les tasques han estat repartides com ha predit el solucionador algebraic. Amb aquest experiment es pot veure com l'edge té el seu propi planificador i quan se satura amb les tasques que li arriben, s'ajuda del Núvol per poder complir el període establert. A més, els dispositius de la capa IoT són capaços d'aprofitar-se de la infraestructura per distribuir les tasques i d'aquesta manera complir amb el període de generació i no penalitzar l'execució del programa si no pogués executar quan toca totes les tasques generades.

Període				720
Dispositiu	rp	CPU	cost	tExec
IoT-1	7200	4	0.4	4
IoT-2	5400	4	0.4	4
IoT-3	4800	4	0.2	4
edge	-	4	0	0.2
Cloud	-	-	2	0.4

Taula 4.3: Configuració exemple IoT - edge - Cloud

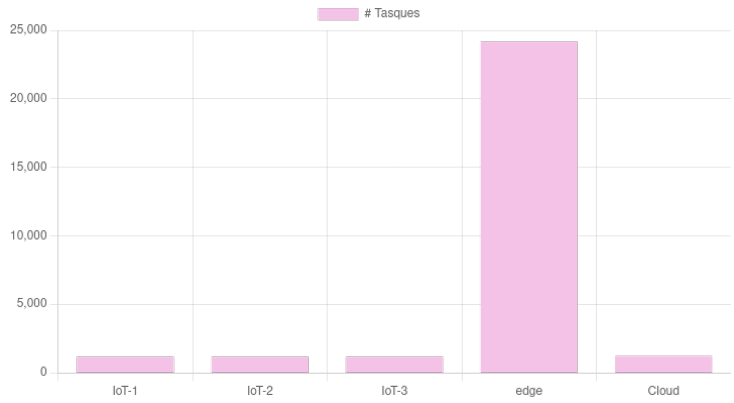


Figura 4.34: Tasques executades a cada dispositiu de la configuració IoT-edge-Cloud.
Font: Elaboració pròpia

5 Informe de sostenibilitat

5.1 Dimensió ambiental

Quant a la dimensió ambiental, aquest projecte pretén desenvolupar planificadors eficients destinats a unes infraestructures específiques a les quals actualment s'utilitzaven planificadors genèrics amb menys rendiment i, per tant, més consum energètic i econòmic. Aquest és el principal motiu, per al qual s'ha optat per desenvolupar noves estratègies en comptes d'utilitzar les ja existents.

D'una banda, aquest projecte no té un impacte ambiental durant la fase de desenvolupament d'aquest. L'únic impacte ambiental durant la fase de desenvolupament és el cost energètic d'utilitzar el maquinari per fer el desenvolupament i proves. Si es vol quantificar el cost ambiental, es pot fer agafant les hores previstes de desenvolupament, juntament amb el consum de cada maquinari utilitzat per saber el consum energètic. Es podria intentar reduir algun del maquinari utilitzat per fer les avaluacions, però això implicaria que no s'hauria provat els planificadors en un entorn real com el que s'utilitzarà durant la seva vida útil. Això, podria comportar no identificar comportaments inesperats en alguna de les polítiques i, per tant, que no funcionés com s'esperava.

D'altra banda, el projecte de COMPSs es pot aplicar a molts àmbits diferents, això complica estimar la dimensió ambiental del projecte. Per exemple, si COMPSs s'utilitzés en un àmbit com l'agricultura i els planificadors desenvolupats ajudessin a reduir el consum de gasolina o de pesticides en un cas hipotètic. Aquest és el motiu que dificulta definir de forma concreta l'abast ambiental del projecte.

Per acabar, si es tornés a fer el projecte, no crec que es pugui reduir el consum energètic, ja que s'han utilitzat el mínim de recursos per poder desenvolupar i testejar els planificadors.

Un altre punt a comentar, és el seu impacte durant la seva vida útil, aquest projecte permetrà reduir l'ús de recursos i globalment, reduir la petjada ecològica. El fet de disposar de polítiques de planificació eficients en entorns de Compute Contínuum permetrà reduir el temps de planificació i, per tant, el temps d'execució cosa que suposarà un ús més eficient dels recursos que s'estiguin utilitzant per a l'aplicació concreta. Com s'ha esmentat prèviament, és molt difícil predir quins recursos s'utilitzaran durant el projecte per culpa de no tenir un usuari final concret sinó que depèn de l'ús que en faci l'usuari que decideixi utilitzat COMPSs. Per exemple es pot utilitzar

en ramaderia, identificar poblacions d'ocells, en ciutats per sistemes de trànsit...

5.2 Dimensió econòmica

Quant a la dimensió econòmica, el pressupost que s'ha dut a terme, era el primer de l'autor del treball i s'ha intentat que els càlculs fossin el més real possible. A més, aquest projecte forma part d'un projecte més gran que és COMPSs que es duu a terme al BSC on aquest posa part del pressupost que es destina en maquinari. Crec que el pressupost no és desorbitat, ja que implicarà una millora quant a la disminució de temps d'execució i, per tant, és una inversió per a la llarga reduir l'impacte econòmic i energètic de les aplicacions que utilitzin aquests planificadors.

Tot i que el cost final no s'hagi ajustat del tot al cost inicial, ja que s'ha afegit maquinari per provar els planificadors en un entorn real i no un teòric. S'ha reutilitzat maquinari que es tenia disponible i no ha suposat una inversió extra. A part d'això, el pressuposat s'ha ajustat correctament i no hi ha hagut desviacions a part d'alguna que ja s'havia previst durant el desenvolupament del pressupost.

Cal afegir, que COMPSs és un projecte que pot tenir aplicacions a diferents àmbits i que les entitats que facin ús dels planificadors fets en aquest treball poden tenir diferents beneficis econòmics respecte a planificadors no adaptats a les infraestructures Compute Contínuum. Per tant, no es pot fer cap estimació del cost que tindrà el projecte durant la seva vida útil. A més, el cost de manteniment serà pràcticament nul gràcies la modularitat del projecte de COMPSs i dels planificadors en si. Es poden fer modificacions en altres polítiques o en el runtime de COMPSs sense que afecti les polítiques implementades.

5.3 Dimensió social

Quant a la dimensió social, la realització d'aquest projecte m'aportarà coneixements de com funció un nou món que ha vingut per quedar-se com és el Compute Contínuum, a més de descobrir COMPSs i contribuir a la millor d'aquest model de programació. És una bona oportunitat per profunditzar els coneixements adquirits durant el Grau en sistemes distribuïts i aplicacions paral·leles.

Com s'ha comentat prèviament, COMPSs té molts àmbits d'ús i per això és difícil acotar quines són les millores de caràcter social que pot tenir el desenvolupament d'aquests planificadors. Pel que fa al runtime, permetrà disposar de nous planificadors per obtenir resultats en menys temps si s'utilitzen en els entorns descrits en aquest treball.

Tanmateix, es pot afirmar que no hi pot haver cap col·lectiu que es vegi perjudicat causa del projecte perquè tothom podrà fer ús dels planificadors vist que no es necessita cap coneixement extra a part dels necessaris per utilitzar COMPSs.

Un altre punt a comentar és la utilitat d'aquest projecte s'ha anat remarcant en diferents apartats com a la contextualització (vegeu secció 1.1). És la manca d'estratègies que s'adaptin al nou entorn del Compute Contínuum. Per tant, el desenvolupament dels planificadors descrits al llarg d'aquest document soluciona aquest problema, ja que els usuaris que vulguin utilitzar COMPSs en aquest entorn disposaran de polítiques de planificació eficients. Com també podran escollir una de les estratègies plantejades i, d'aquesta manera, és l'usuari que té el poder de decisió sobre què vol utilitzar, en conseqüència, no suposarà cap mena de dependència de l'usuari envers el contingut desenvolupat en aquest projecte.

Per concloure, m'agradaria parlar de la dimensió de gènere del projecte. S'ha intentat que es compongui per un 50% de dones i d'homes. L'equip està compost per una dona, la cap de projecte, i dos homes, l'investigador sènior i l'enginyer. A part d'això no té cap impacte, ja que es desenvolupen planificadors per a un model de programació i no té rellevància el gènere de l'usuari final.

6 Conclusions

6.1 Resultats tècnics

S'ha implementat tres estratègies de planificació, una basada en FIFO, una altra en Backpressure i la tercera en Teoria de Jocs. Cadascuna de les tres estratègies desenvolupades funciona correctament i com s'ha estudiat en la secció d'avaluació (vegeu secció 4.1) hem vist que hi ha polítiques que funcionen millor en uns casos i polítiques que funcionen millor en altres. Concretament, la política FIFO funciona per a qualsevol classe d'infraestructura, des de supercomputadors com el MareNostrum 4, fins a dispositius IoT o fins i tot el Núvol. És una política ràpida i no es necessiten requisits per utilitzar-la. L'únic inconvenient del FIFO és que no té en compte la localitat de les dades i en programes amb dades grans pot ser menys eficient que altres polítiques que sí que tinguin en compte la localitat.

La segona política que s'ha desenvolupat és la de backpressure. Aquesta s'ha vist que quan hi ha un nombre elevat de tasques el seu rendiment empitjora bastant, però, d'altra banda, et permet anar distribuint la càrrega de tasques als nodes d'execució amb més recursos disponibles. A més, aquest planificador té en compte la localitat de les dades i en certs programes amb una gran quantitat de paràmetres pot ser d'utilitat. Tot i això, aquesta estratègia té un problema de rendiment, ja que se li acumulen moltes tasques i això pot provocar un comportament més lent i, per tant, que les tasques acabin enviant-se fora del recurs local per la penalització que suposa tenir tasques acumulades.

La tercera i última estratègia de planificació ha estat basada en teoria de jocs, aquesta política és molt ràpida i molt bona en casos concrets. Per exemple, si tens una generació constant de tasques i tens únicament dos recursos, el local i un remot compartit amb altres recursos, aquest planificador és capaç de planificar les tasques a ambdós recursos sense saturar el recurs extern però donant-li prioritat. Els inconvenients d'aquest planificador és que necessita uns requisits concrets per poder funcionar com tenir únicament dos recursos i que la generació de tasques no sigui variable. Tot i això, aquest planificador encaixa perfectament amb la capa IoT com s'havia dissenyat.

A més, s'han provat tots tres planificadors en un entorn IoT-edge-Cloud, és a dir, el FIFO a la capa del Cloud, el backpressure a la capa edge i el de teoria de jocs a la capa IoT. S'ha comprovat com utilitzar cada planificador a la capa que originalment s'havia dissenyat ha donat bons resultats i les tasques s'han repartit entre les diferents capes sense saturar-ne cap. En

resum, en finalitzar aquest treball COMPSs disposa d'estratègies de planificació per a entorns de *Compute Contínuum* i s'ha comprovat el correcte funcionament d'aquests planificadors en tests unitaris, avaluant el temps de planificar les tasques segons unes variables i, finalment s'ha fet un cas sintètic de laboratori dels tres planificadors.

6.2 Desviacions del projecte

En aquest apartat es fa una valoració de com s'ha seguit la planificació que s'ha fet del projecte, si s'ha complert la planificació temporal, quins imprevists o riscs s'han complert i els canvis que s'han produït durant el transcurs del projecte. Primerament, analitzarem si s'ha pogut desenvolupar i avaluar tot el que estava previst. Es volia desenvolupar tres planificadors i per cada planificador analitzar quins paràmetres afecten el temps de planificar tasques, generar tests unitaris per comprovar el correcte funcionament, i finalment desplegar una infraestructura amb les tres capes IoT-edge-Cloud i mirar com es comporten els tres planificadors. En el temps que s'ha disposat per la realització del treball s'ha pogut desenvolupar els tres planificadors FIFO, Backpressure i Teoria de Jocs. També s'ha fet l'avaluació de quins paràmetres afecten quan s'han de planificar les tasques per als tres planificadors. Així mateix, s'ha desenvolupat tests unitaris únicament per a la política FIFO perquè hi ha hagut falta de temps i s'ha prioritzat poder fer l'avaluació dels tres planificadors en un entorn IoT-edge-Cloud.

Pel que fa als imprevists i riscs, durant la fase de disseny de la política de backpressure, s'ha detectat la necessitat d'informació detallada del *profile* que disposa cada *RS* per poder saber el temps que ha estat en espera una tasca, quant de temps ha tardat a executar-se. Aquest fet ha provocat que s'hagi de modificar el runtime per afegir aquesta informació a la classe *profile* i com estava previst aquesta modificació ha estat implementada per un investigador sènior del grup de recerca. Un altre possible risc previst que ha sorgit ha estat durant la fase de disseny de la política basada en teoria de jocs on el plantejament inicial que es va fer per a la capa del servidor edge no acabava d'encaixar amb el runtime ni amb la capa d'edge. A més, es va veure que la política de teoria de jocs podia encaixar millor a la capa de IoT perquè era més fàcil identificar els jugadors i l'estratègia que pot optar cada dispositiu IoT. Per tant, es va decidir baixar al nivell de IoT l'estratègia de teoria de jocs i pujar al nivell d'edge la política de backpressure. Un cop fet aquest canvi sí que va ser possible dissenyar una política que encaixés amb l'arquitectura de COMPSs i es basi en teoria de jocs.

Pel que fa a l'àmbit econòmic no hi ha hagut desviacions pel que fa al pressupost. Els riscs i imprevists que es mencionen es van considerar en les partides de contingència i riscs.

6.3 Pròxims passos

Els pròxims passos per a aquest projecte serien millorar el planificador de backpressure per augmentar el rendiment, analitzar i implementar una millor manera per calcular la penalització que se li assigna a cada recurs. Una manera de millorar el càlcul de la penalització pot ser replantejar el mètode que hi ha actualment, buscar una funció matemàtica més precisa o, fins i tot, utilitzar *Machine Learning* per trobar la funció òptima per cada recurs.

Un altre punt que es podria millorar és el planificador basat en teoria de jocs. Actualment, es calcula el percentatge abans d'aixecar l'agent i no es pot modificar aquest percentatge en temps d'execució. Una millora seria poder afegir al runtime el solucionador algebraic i que ell mateix pugui calcular el percentatge de tasques que ha d'enviar en temps d'execució. A més, això permetria poder-se adaptar a canvis que es produeixin en la infraestructura i fer el repartiment de tasques més eficient.

També, s'haurien de dissenyar i implementar els tests unitaris que no s'han pogut desenvolupar pels planificadors de backpressure i teoria de jocs. Fer aquests tests unitaris permet tenir una eina de verificació per quan es facin modificacions al planificador base o al runtime i que aquests planificadors continuïn funcionant com s'espera.

Un altre treball futur que es podria fer seria provar com es comporten aquests planificadors utilitzant una aplicació real i no un cas sintètic com s'ha utilitzat en aquest projecte. Depenent del cas d'ús que es vulgui utilitzar, es podran utilitzar diferents dispositius IoT i veure com es comporten que és una cosa que en aquest treball no s'ha pogut observar.

6.4 Competències tècniques

En aquesta secció analitzarem com s'han treballat les competències tècniques marcades quan es va inscriure aquest projecte com a TFG.

La primera competència que es va marcar és la *CEC2.1*, i consisteix a *Analitzar, avaluar, seleccionar i configurar plataformes hardware per al desenvolupament i l'execució d'aplicacions i serveis informàtics.* Aquesta s'ha tractat en l'avaluació que s'ha fet dels planificadors utilitzant les Raspberry Pi, la Jetson i Azure com a Núvols. En cadascun d'aquests dispositius s'ha configurat un entorn amb COMPSs i els planificadors desenvolupats perquè es puguin executar aplicacions utilitzant la infraestructura descrita a l'inici d'aquest treball, IoT-edge-Cloud.

La segona competència que s'ha treballat és la *CEC2.2*, que consisteix a *Programar considerant l'arquitectura hardware, tant en assemblador com en alt nivell.* Això s'ha treballat quan s'ha dissenyat els planificadors s'ha analitzat les característiques de cadascuna de les capes, fet que ha comportat que es dissenyin planificadors conscients de l'arquitectura de cada cap. Per exemple,

el de la capa de IoT, el de teoria de jocs, s'ha pensat que fos un planificador que requerís poca capacitat de càlcul, ja que la poca capacitat de càlcul és una de les característiques dels dispositius IoT. El mateix s'ha fet amb el Núvol, és una capa amb molts recursos i també allunyada de la capa a on es generen les tasques. Això comporta que sigui important tenir un planificador àgil que pugui planificar gran quantitat de tasques a molts recursos i que no empitjori la latència que té enviar una tasca al Núvol.

Finalment, la tercera competència que s'ha assignat al treball és la *CEC2.4* i tracta de *Dissenyar i implementar software de sistema i de comunicacions.* Tot el treball està sobre el model de programació de COMPSs i amb l'objectiu d'afegir-hi noves estratègies de planificació. Per tant, tots els planificadors implementats en aquest treball són per al model de programació de COMPSs.

A Taula de conceptes

- ***Internet of Things (IoT)***
  Dispositius de poca capacitat de càlcul, com sensors o microcontroladors. Utilitzen internet per intercanviar dades amb altres dispositius i sistemes com per exemple el Núvol. Aquests dispositius acostumen a ser d'un usuari.

- ***Edge***
  Servidor que té més capacitat de càlcul i emmagatzematge que un dispositiu IoT i és proper a l'usuari. Acostumen a ser propietat d'empreses de telecomunicacions o empreses capaces de desplegar infraestructures.

- ***Cloud***
  Infraestructura de sistemes informàtics sobretot d'emmagatzematge i càlcul. On l'usuari pot sol·licitar sota demanda els recursos que necessitarà i es paga per l'ús que es fa d'aquesta infraestructura. Aquestes infraestructures acostumen a estar allunyades de l'usuari.

- ***Compute Continuum***
  Infraestructura que es coneix tot els components que la formen. A més, afegeix capes entre els dispositius IoT i el Núvol.

- ***Workflow***
  Conjunt de tasques que depenen entre si formant un flux d'execució. Les tasques es relacionen entre si mitjançant les dependències de dades.

- ***Core Element (CE)***
  Component d'una aplicació que defineix el comportament d'una acció. Aquesta part pot tenir diferents implementacions. Un exemple seria un sort com a CE sense especificar l'algorisme.

- ***Implementació***
  Mètode específic que indica com s'executarà un CE. Per exemple, si tenim un sort com a CE, una implementació seria un *MergeSort* o un *BubbleSort*.

- ***Tasca***
  Component asíncron que donades unes dades d'entrada genera unes dades de sortida. Una tasca correspon a una instància d'un CE.

- ***Programming Model***

Màquina abstracta que, per una banda, defineix una manera com l'usuari interactua amb ella i, per altra banda, defineix un model d'execució que indica com s'executarà el programari.

- ***Runtime System/Engine***
  Part del programari que s'executa paral·lelament amb l'aplicació de l'usuari. Determina el comportament que tindrà el programming model la màquina abstracta.
- ***Planificador***
  Àrbitre que, seguint una política o estratègia, decideix l'ordre en què s'executaran les tasques i a quin dels recursos anirà.
- ***Locality Scheduler***
  Planificador per defecte de COMPSs que planifica les tasques tenint en compte la localitat temporal de les dades de les tasques i, a continuació l'ordre FIFO d'aquestes.
- ***Slurm***
  És un sistema de planificació de jobs i gestió de clúster utilitzat a MareNostrum 4.
- ***Job (Slurm)***
  Tasca de Slurm que s'utilitza per llançar un programa a un clúster (MN4).
- ***Job (COMPSs)***
  Cada cop que una tasca s'envia a un recurs.
- ***Extrae***
  Eina desenvolupada pel BSC que serveix per instrumentar codi. S'utilitza per extreure els esdeveniments i generar traces per visualitzar-les a Paraver.
- ***Traça***
  Representació gràfica que mostra els esdeveniments que tenen lloc a cada recurs al llarg del temps d'execució d'un programa.
- ***Comma Separated Values (csv)***
  És un tipus de document de format obert que s'utilitza per representar valors en format taula d'una manera senzilla. S'acostumen a separar els valors de les columnes en comes.
- ***Java Development Kit (JDK)***
  És el programari que proveeix les eines de desenvolupament necessàries per crear programes escrits en Java.

Bibliografia

[1] *COMP Superscalar | BSC-CNS*. URL: https://www.bsc.es/ca/research-and-development/software-and-apps/software-list/comp-superscalar (cons. 17-03-2022).

[2] *The New Compute Continuum is Powering the Future of IoT – Arm®*. URL: https://www.arm.com/resources/white-paper/compute-continuum (cons. 17-03-2022).

[3] The OpenFog Consortium. "OpenFog Architecture Overview". A: *OpenFogConsortium* February (2016), pàg. 1-35. URL: www.OpenFogConsortium.org.

[4] Francesc Lordan, Daniele Lezzi i Rosa M. Badia. "Colony: Parallel Functions as a Service on the Cloud-Edge Continuum". A: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12820 LNCS (2021), pàg. 269-284. ISSN: 16113349. DOI: 10.1007/978-3-030-85665-6_17.

[5] Tie Qiu, Ruixuan Qiao i Dapeng Oliver Wu. "EABS: An event-aware backpressure scheduling scheme for emergency internet of things". A: *IEEE Transactions on Mobile Computing* 17.1 (2018). ISSN: 15580660. DOI: 10.1109/TMC.2017.2702670.

[6] Sarhad Arisdakessian et al. "FoGMatch: An Intelligent Multi-Criteria IoT-Fog Scheduling Approach Using Game Theory". A: *IEEE/ACM Transactions on Networking* 28.4 (2020), pàg. 1779-1789. ISSN: 15582566. DOI: 10.1109/TNET.2020.2994015.

[7] *Portàtil Dell Latitude 7420*. URL: https://www.dell.com/es-es/work/shop/port%C3%A1tiles-dell/latitude-7420/spd/latitude-14-7420-2-in-1-laptop/s0591742014w11dgres (cons. 11-03-2022).

[8] Barcelona Supercomputing Center. *MareNostrum | BSC-CNS*. 2021. URL: https://www.bsc.es/marenostrum/marenostrum (cons. 16-03-2022).

[9] *Agile software development - Wikipedia*. URL: https://en.wikipedia.org/wiki/Agile_software_development (cons. 17-03-2022).

[10] *Kanban - Wikipedia*. URL: https://en.wikipedia.org/wiki/Kanban (cons. 17-03-2022).

[11] *Git*. URL: https://git-scm.com/ (cons. 17-03-2022).

[12] *What is CI/CD?* URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd (cons. 17-03-2022).

[13] *JUnit 5*. URL: https://junit.org/junit5/ (cons. 18-03-2022).

[14] *Overleaf, Online LaTeX Editor*. URL: https://www.overleaf.com/ (cons. 17-03-2022).

[15] *IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains*. URL: https://www.jetbrains.com/idea/ (cons. 17-03-2022).

[16] *Video Conferencing, Cloud Phone, Webinars, Chat, Virtual Events | Zoom.* URL: `https://zoom.us/` (cons. 17-03-2022).

[17] *Skype | Stay connected with free video calls worldwide.* URL: `https://www.skype.com/en/` (cons. 17-03-2022).

[18] *Vim, the editor.* URL: `https://www.vim.org/` (cons. 17-03-2022).

[19] *Graphviz.* URL: `https://graphviz.org/` (cons. 17-03-2022).

[20] *Trello.* URL: `https://trello.com/` (cons. 17-03-2022).

[21] *Iterate faster, innovate together | GitLab.* URL: `https://about.gitlab.com/` (cons. 21-03-2022).

[22] *GanttProject - Free Project Management Application.* URL: `https://www.ganttproject.biz/` (cons. 17-03-2022).

[23] *Paraver: a flexible performance analysis tool | BSC-Tools.* URL: `https://tools.bsc.es/paraver` (cons. 16-06-2022).

[24] *Calculadora salarial.* URL: `https://guiasalarial.hays.es/empresa/calculadora-salarial` (cons. 11-03-2022).

[25] *Math – Commons Math: The Apache Commons Mathematics Library.* URL: `https://commons.apache.org/proper/commons-math/index.html` (cons. 30-05-2022).

[26] *Random forest - Wikipedia.* URL: `https://en.wikipedia.org/wiki/Random_forest` (cons. 19-06-2022).