Cloudpaging per contenidors d'aplicacions

Treball de Final de Grau

Autor:
Xavier Pijuan Garangou
Director:
Juan Jose Costa Prats
Departament d'Arquitectura de Computadors
Data de defensa:
24 d'abril de 2018

Universitat Politècnica de Cataluña (UPC) – BarcelonaTech
Facultat d'Informàtica de Barcelona (FIB)
Grau en Enginyeria Informàtica
Especialitat d'Enginyeria de computadors

Resum

L'èxit dels serveis de computació en el núvol ens ensenya que la flexibilitat és un dels recursos més valuosos dels sistemes informàtics. Els contenidors d'aplicacions són una eina per aconseguir aquesta flexibilitat en el desplegament d'aplicacions, però la seva mida pot alentir la seva transferència, cosa que pot ser un problema per als clients que necessiten desplegar aplicacions de manera instantània.

Per mitigar aquest problema, aquest projecte presenta el disseny i la implementació d'un sistema *software* per sistemes operatius basats en *Linux,* que permet efectuar tant la migració com el desplegament de contenidors entre ordinadors de manera pràcticament instantània.

El sistema utilitza l'algoritme de *cloudpaging,* anomenat així per la seva similitud amb el sistema de paginació (*paging*) que fan servir els sistemes operatius per gestionar la memòria, consistent en dividir els arxius en blocs de mida uniforme, i copiar-ne cada un a la memòria local el primer cop que es demana.

El nostre sistema, per tant, presenta un sistema d'arxius virtual a l'aplicació, que obté les parts necessàries dels arxius del servidor remot en temps real. Això al seu torn permet executar les aplicacions al mateix temps que es descarreguen. Es categoritza doncs com un sistema d'*application streaming*.

Com que treballa a nivell d'arxius, el nostre sistema és universal i no requereix cap modificació en les aplicacions a executar.

Com a complement al sistema principal de *cloudpaging*, hem creat a més un sistema que permet interceptar les operacions de lectura i escriptura que efectuï qualsevol altre programa, amb l'objectiu de recollir dades estadístiques d'aquestes.

Resumen

El éxito de los servicios de computación en la nube nos enseña que la flexibilidad es uno de los recursos más valiosos de los sistemas informáticos. Los contenedores de aplicaciones son una herramienta para conseguir esta flexibilidad en el despliegue de aplicaciones, pero su tamaño puede ralentizar su transferencia, cosa que puede suponer un problema para los clientes que necesitan desplegar aplicaciones de manera instantánea.

Para mitigar este problema, este proyecto presenta el diseño y la implementación de un sistema *software* para sistemas operativos basados en *Linux,* que permite efectuar tanto la migración como el despliegue de contenedores entre ordenadores de manera prácticamente instantánea.

El sistema utiliza el algoritmo de *cloudpaging,* llamado así por su similitud con el sistema de paginación (*paging*) que utilizan los sistemas operativos para gestionar la memoria, consistente en dividir los archivos en bloques de tamaño uniforme, y copiar cada uno a la memoria local la primera vez que se solicita.

Nuestro sistema, por tanto, presenta un sistema de archivos virtual a la aplicación, que obtiene las partes necesarias de los archivos del servidor remoto en tiempo real. Esto por su parte permite ejecutar las aplicaciones al mismo tiempo que se descargan. Se categoriza por tanto como un sistema de *application streaming*.

Como trabaja a nivel de archivo, nuestro sistema es universal y no requiere ninguna modificación en las aplicaciones a ejecutar.

Como complemento al sistema principal de *cloudpaging*, hemos creado además otro sistema que permite interceptar las operaciones de lectura y escritura que efectúe cualquier otro programa, con el objetivo de recoger datos estadísticos.

Abstract

The success of cloud computing services teaches us that flexibility is one of the most valuable resources in computer systems. Application containers are a tool to obtain this flexibility in application deployment, but their size can slow down their transfer, which can pose a problem for users that need to deploy applications on very short notice.

To tackle this problem, our project presents the design and implementation of a software system for Linux-based operating systems, that allows both migration and deployment of containers between computers almost instantaneously.

This system uses the *cloudpaging* algorithm, given that name because of its similarity with the paging algorithm used by operating systems to manage memory, which consists in dividing files into blocks of constant size, and copying each one to local memory the first time it's needed.

Our system, therefore, presents a virtual file system to the application, that obtains the necessary parts of the files from the remote server in real time. This allows executing applications at the same time they are being downloaded. It's therefore categorized as an *application streaming* system.

Since it works at file level, our system is universal and does not require any modification to the applications being executed.

To complement the main *cloudpaging* system, we also created a system that can intercept all read and write operations that another program makes, with the purpose of gathering statistics about them.

Índex

1. Introducció i objectius 1
1.1 Context 1
1.2 Contenidors d'aplicacions 2
1.3 Memòria virtual i paginació (paging) 2
1.4 Problema 3
1.5 Solució: *application streaming* 4
1.6 Estat de l'art 5
1.7 Objectius 6
2 Gestió del projecte i abast 7
2.1 Metodologia 7
2.2 Planificació i fases del projecte 7
2.3 Estimació del temps 8
2.4 Mètodes de validació i pla d'acció 9
2.5 Eines de desenvolupament 9
2.6 Gestió econòmica 10
3 Sostenibilitat 14
3.1 Estudi d'impacte ambiental 14
3.2 Estudi d'impacte econòmic 15
3.3 Social 16
3.4 Avaluació de sostenibilitat 17
4 Els sistemes d'arxius en Linux 18
4.1 El nucli Linux 18
4.2 Operacions d'arxiu en Linux 18
4.3 Arxius especials 19
4.4 Mòduls i muntatge 19
4.5 La interfície Virtual File System 20
4.6 FUSE (Filesystem in Userspace) 21
5 Disseny del sistema 22
5.1 Enfocament del disseny: l'algoritme de cloudpaging 22
5.2 Disseny 22
5.3 Estructures de dades 23
6 Sistema d'anàlisi d'us d'arxius 24

6.1 Disseny 24
7 Desenvolupament 25
7.1 Part comuna: sistemes d'arxius 25
7.2 Sistema de cloudpaging 28
8 Resultats 33
8.1 Sistema de cloudpaging 33
8.2 Sistema d'anàlisi d'us d'arxius 33
8.3 Gestió econòmica i temporal 36
9 Conclusions 38
9.1 Treball futur 38
10 Bibliografia 40
11 Annex 1: Diagrama de Gantt 43

1. Introducció i objectius

En aquest capítol explicarem el context (conceptes relacionats), el problema que volem solucionar, l'estat de l'art i definirem l'objectiu del nostre projecte.

1.1 Context

En aquest apartat explicarem els conceptes importants necessaris per entendre els objectius del projecte, és a dir, el context.

El concepte més important són els contenidors d'aplicacions. Per poder entendre els contenidors, però, introduirem primer el concepte de computació en el núvol.

Computació en el núvol

En l'última dècada s'ha experimentat un creixement massiu en la popularitat del *cloud computing*, o computació en el núvol.

El "núvol" consisteix, essencialment, en un conjunt d'ordinadors, generalment gestionats per una empresa dedicada, que proveeix un servei informàtic (emmagatzematge de dades o execució de programes), al que l'usuari pot accedir com una sola entitat, sense haver-se de preocupar del seu número o de la seva estructura.

Aquesta separació entre ordinadors físics i serveis proporciona una enorme flexibilitat, tant a l'usuari com al proveïdor. L'usuari pot expandir el seu programa d'una instància a centenars de manera immediata, o guardar dades sense haver de preocupar-se de realitzar còpies redundants per si falla un disc, i el proveïdor pot apagar ordinadors per realitzar manteniment sense que l'usuari noti cap canvi.

Exemples de serveis que es proporcionen habitualment en el núvol són:

- Servidors privats virtuals (VPS): sistemes que permeten disposar d'un "ordinador virtual" en el que l'usuari pot executar programes
- Emmagatzematge d'arxius: pot ser orientat a usuaris (com *Google Drive* o *Dropbox*) o a programadors (com *Amazon S3* o *Azure Storage*)
- Bases de dades SQL
- "*Game streaming*": sistemes optimitzats per l'execució de videojocs, on la imatge es transmet amb una latència molt baixa a l'usuari, de manera que pot jugar com si l'estés executant en el seu propi ordinador

Com a exemple de la importància i el creixement de la computació en el núvol, podem mencionar que el proveïdor Amazon Web Services, propietat de l'empresa Amazon, ha tingut un increment anual mitjà en els seus ingressos del 65% entre 2011 i 2016 [1], i el 2017 ha representat el negoci més profitós de l'empresa. La seva quota en el mercat de proveïdors de cloud s'estima en el 30% [2].

1.2 Contenidors d'aplicacions

Per implementar aquesta flexibilitat del núvol, es requereix crear una capa d'abstracció, és a dir, un sistema informàtic que pugui moure aquests recursos d'un servidor a un altre. Els contenidors d'aplicacions són un d'aquests sistemes.

Un contenidor d'aplicacions és un sistema que permet executar *software* en un entorn virtual i controlat, independent del sistema operatiu real. Això significa que els únics arxius que l'aplicació pot veure són els que estan en el seu disc virtual, que sol ser un arxiu o carpeta del sistema operatiu amfitrió.

És similar a una màquina virtual, però és molt més eficient, ja que la virtualització es realitza en un nivell més alt. El codi s'executa de manera nativa, i no s'ha d'executar una còpia del sistema operatiu per cada aplicació.

Existeixen moltes tècniques per implementar-los. La majoria de sistemes operatius ja tenen nativament sistemes que permeten aïllar processos, i en particular, els sistemes basats en Unix són molt flexibles amb l'estructura del sistema d'arxius. Per exemple, en el Linux existeix la trucada al sistema *chroot*, que permet modificar el sistema d'arxius que veu un procés particular. En general, els sistemes de contenidors d'aplicacions són una extensió d'aquests sistemes.

Existeixen moltes implementacions. Per exemple, en el Linux trobem *Docker, LXC, Linux-VServer* i *OpenVZ*. En aquest projecte, no ens centrarem en cap d'elles en concret, sinó que desenvoluparem un sistema genèric que podrà funcionar en combinació amb la majoria de sistemes de contenidors.

Els avantatges de fer servir contenidors són molts:

- Augmenten la seguretat, aïllant cada programa de la resta del sistema
- Faciliten l'execució de programes. Com que cada contenidor inclou tots els components necessaris, com llibreries i programes auxiliars, no hi ha problemes de compatibilitat en distribucions diferents de Linux
- Faciliten la instal·lació i desinstal·lació de programes, ja que només s'ha de copiar o esborrar el contenidor i no cal copiar arxius a carpetes diferents del sistema ni executar scripts d'instal·lació (cal dir que això no és mèrit dels contenidors sinó desmèrit del Linux)
- Permeten "moure" una aplicació d'un ordinador a un altre, copiant el seu estat complet perquè es continuï executant en el nou sense que l'aplicació noti cap canvi

1.3 Memòria virtual i paginació (paging)

Aquesta idea és necessària per entendre el concepte de *cloudpaging* que veurem més endavant.

Per gestionar la memòria de forma eficient, els ordinadors moderns utilitzen un sistema anomenat memòria virtual, on totes les lectures i escriptures a memòria d'un

programa es tradueixen en temps real a posicions diferents de la memòria física, amb l'ajuda d'un dispositiu en el processador anomenat unitat de gestió de memòria (més conegut com a *Memory Management Unit* o *MMU*). Per fer-ho, divideixen l'espai de memòria en blocs de la mateixa mida (normalment *4 kB*), anomenats "pàgines".

Una de les coses que la memòria virtual permet és moure les pàgines de la memòria al disc dur i viceversa de manera dinàmica. Així, si una part de la memòria no ha estat usada en un cert període de temps, es pot moure al disc dur i aprofitar millor l'espai de la memòria.

Quan un procés intenta llegir una pàgina que no està disponible a la memòria, el sistema operatiu atura el procés, copia la pàgina a la memòria, i reprèn el procés quan la còpia s'ha completat. D'aquesta manera, el sistema és totalment transparent per al procés, que veu la memòria com si sempre hi fossin totes les dades.

1.4 Problema

Els contenidors d'aplicacions poden ser molt grans. Això significa que copiar-los d'un ordinador a un altre pot tardar una estona, encara que es tingui una bona amplada de banda.

Això pot ser un problema per la computació en el núvol, ja que redueix la flexibilitat de l'usuari. En particular, podem identificar dos escenaris que requereixen copiar un contenidor entre ordinadors en molt poc temps:

1. Desplegar un servidor. En molts casos una empresa que proveeix un servei en línia tindrà una imatge preconfigurada del servidor que utilitza, i desplegarà més o menys imatges segons la demanda d'usuaris.

   Per exemple, imaginem una empresa que hagi publicat un videojoc en línia multijugador. En la majoria de casos, aquests videojocs necessiten un programa servidor operat per l'empresa al qual es connecten els clients. A més, per reduir la latència que experimenta l'usuari, necessiten que el servidor estigui geogràficament tan a prop de l'usuari com sigui possible.

   Si el proveïdor de cloud pot crear servidors nous en pocs segons, podrà tenir sempre exactament el nombre d'instàncies que necessiti per cobrir la demanda actual. Si no, n'haurà de tenir un nombre extra a l'espera a cada ubicació geogràfica per si es connecta un usuari. Això és ineficient i augmenta els costos.

2. Moure una aplicació activa d'un ordinador a un altre.

   Com hem explicat anteriorment, és important per al proveïdor del cloud poder apagar els ordinadors per realitzar manteniment o estalviar energia. Però si l'aplicació que s'està executant està sent utilitzada per un usuari, aquest es quedarà sense servei durant la migració. En la majoria de contextos, una aturada de més de diversos segons és inacceptable.

1.5 Solució: *application streaming*

Una manera de mitigar aquest problema és enviant les dades del contenidor al mateix temps que s'executen. D'aquesta manera, el programa pot començar a executar-se de manera quasi immediata, i només s'aturarà durant períodes curts quan intenti llegir dades que encara no estiguin disponibles.

Aquesta idea s'anomena *application streaming*, o enviament d'aplicacions en temps real [3] i és quasi tan antiga com la mateixa idea de les xarxes d'ordinador. En l'actualitat es sol combinar amb màquines virtuals o contenidors, però no és estrictament necessari.

El concepte és idèntic al *video streaming* (transmissió de vídeo) que tots utilitzem en serveis com YouTube o Netflix, on el client va reproduint i rebent les dades al mateix temps.

És important no confondre aquest concepte amb els sistemes d'escriptori remot, que a vegades fan servir també el terme "streaming". En l'escriptori remot, l'usuari accedeix a un altre ordinador i interactua amb programes en temps real, però el codi de l'aplicació no es transmet, només la imatge de la pantalla i les accions de l'usuari.

Eficàcia

En el cas del vídeo, la reproducció es podrà fer sense pauses sempre que la velocitat de descàrrega sigui superior a la velocitat de reproducció, ja que es reprodueix de forma lineal.

En el cas del software, no tenim aquesta sort, ja que no és possible predir quina part del programa s'executarà a continuació (pot dependre de les accions de l'usuari o de les dades rebudes). Per tant, hi haurà inevitablement algunes pauses quan el programa intenti accedir a parts encara no carregades, però si la latència de la xarxa és suficientment baixa i l'amplada de banda és suficientment gran, aquestes podran ser imperceptibles.

Ens ajudem de l'anomenat principi de Pareto, també conegut com la "llei 80-20". El principi de Pareto és una regla empírica que ens diu que el 80% dels efectes provenen del 20% de les causes [4]. El nom prové de l'economista Vilfredo Pareto, que va observar el 1896 que el 80% de la terra a Itàlia era propietat del 20% de la població, i que el 80% dels pèsols del seu jardí venien del 20% de les mongetes.

Es pot observar empíricament en camps molt variats:

- El 20% dels pacients del sistema de salut dels Estats Units utilitzen el 80% dels recursos [5]
- El 20% de la població comet el 80% dels crims [6]
- El 20% dels errors de programació causen el 80% dels problemes d'execució [7]
- La majoria d'usuaris fan servir el 20% de la funcionalitat dels programes [8]

Però el més important per nosaltres és l'observació que el 80% del temps de procés d'un programa ve del 20% del codi [9]. De fet, en optimització de programes a vegades es diu fins i tot que el 90% del temps d'execució ve del 10% del codi [10]. Esperarem, per tant, que la majoria de contenidors d'aplicacions mantinguin aquest patró de lectura exponencial.

Això té sentit si pensem que un dels elements principals que s'utilitzen per crear programes són els bucles (seccions petites del codi que s'executen moltes vegades) i que d'altra banda grans parts del codi sol existir només per gestionar errors o casos especials, i per tant no farà falta la majoria de vegades.

Aquesta distribució significarà que, en executar un contenidor d'aplicacions fent servir *application streaming*, tindrem un període inicial en què el programa carregarà la part important del codi (el "20%"), però després les pauses es faran més curtes i s'aniran fent cada cop menys freqüents.

1.6 Estat de l'art

En aquesta secció farem un anàlisi dels sistemes existents de *application streaming*.

Sistemes comercials

En l'actualitat podem trobar ja molts sistemes comercials que l'implementen, principalment en Windows, incloent:

1. *Citrix Application Streaming*: integrat en el sistema de virtualització *XenApp* [11], tot i que es va eliminar a partir de la versió 7 [12]
2. *Microsoft App-V application streaming* [13]
3. *MicroFocus Desktop Containers* [14]
4. *Numecent Cloudpaging* [15]
5. *Microsoft Office "Click-to-run"*: s'integra en la *suite* de programes Microsoft Office, i s'utilitza per agilitzar la seva distribució digital [16].

Sistemes acadèmics

En l'àmbit acadèmic es poden trobar gran multitud de *papers* descrivint sistemes de *application streaming*. Molts d'ells van ser ideats per permetre executar aplicacions en sistemes mòbils amb una capacitat de memòria molt més limitada que la que trobem en els sistemes mòbils actuals.

Trobem, per exemple, un sistema d'arxius *streaming* per Linux molt similar al resultat del nostre projecte [17], sistemes que operen modificant la memòria del sistema [18] [19], o un sistema que envia classes a la màquina virtual Java a mesura que es requereixen [20]

Podem destacar un sistema publicat en 1996 anomenat "*Liquid Software*" [21], que permetia crear funciones que podien ser distribuïdes fàcilment entre ordinadors, orientat a permetre xarxes que poguessin variar el seu comportament dinàmicament.

Tot i això, cap dels sistemes que hem vist ha estat publicat com a solució software real, sinó que han romàs limitats a la teoria.

Sistemas de contenidors

Alguns sistemes de contenidors per Linux tenen la funció de “Live Migration”, és a dir, migrar contenidors d’un amfitrió a un altre, utilitzant tècniques de *application streaming* per minimitzar el temps d’espera:

- Virtuozzo [22]
- runC [23]

Altres

Algunes plataformes d’aplicacions estan dissenyades incorporant la idea de l’execució en xarxa, per exemple HTML+HTTP. En aquestes plataformes, la càrrega d’arxius de codi o recursos s’efectua en temps d’execució, i el programador les pot controlar de diverses maneres.

Lògicament, però, això només es pot aprofitar en aplicacions que s’hagin desenvolupat per aquestes plataformes. Per tant no és cap ajuda per les organitzacions que disposin ja de programes per plataformes diferents.

Conclusió

Existeixen actualment sistemes comercials per Windows que fan el que volem, però cap que sigui obert i per Linux.

1.7 Objectius

Malgrat tots els sistemes que hem mencionat en l’apartat anterior, no n’hem trobat cap que sigui obert i gratuït, que funcioni en Linux, i que no estigui lligat a cap sistema de contenidors concret. L’objectiu del nostre projecte, doncs, serà crear-ne un.

En altres paraules, el nostre objectiu serà crear un sistema software per Linux que permeti, combinat amb qualsevol sistema de contenidors d’aplicacions, executar els contenidors en un ordinador client, obtenint les dades d’un altre ordinador servidor a mesura que fan falta, i publicar-lo amb llicència GPL.

El sistema estarà dirigit a qualsevol empresa o organització que mantingui sistemes de servidors Linux, i el seu objectiu serà facilitar-los l’administració dels serveis que desitgin executar en aquests. Els principals usuaris serien els proveïdors comercials de serveis *cloud*, que permeten a qualsevol altre client executar programes en els seus ordinadors. Indirectament, se’n beneficiaran els seus clients, que podran obtenir un servei millor i més barat, i finalment els consumidors finals, que potencialment poden ser qualsevol persona que utilitzi qualsevol servei en línia.

2 Gestió del projecte i abast

2.1 Metodologia

Les metodologies populars de desenvolupament com Scrum, Agile, Kanban, etc. estan dissenyades per optimitzar el treball en equip, mentre que el nostre projecte serà desenvolupat per una sola persona.

Per això, per al desenvolupament del programa, hem decidit utilitzar una metodologia inspirada en Scrum, però no idèntica:

- El programa es desenvoluparà en "sprints", o iteracions curtes, per tal que les seves funcionalitats es puguin anar afegint progressivament. Al final de cada iteració es disposarà d'un producte funcional (el codi compila i s'executa correctament), encara que no tingui totes les funcionalitats que vulguem o no estigui optimitzat. Això ens donarà una gran flexibilitat en cas que se'ns acabi el temps, ja que ens permet interrompre el treball en qualsevol moment i obtenir el millor producte possible.
- *Test-driven*: crearem un joc de proves automàtic i l'executarem al final de cada iteració per comprovar que el programa funciona sempre perfectament. Així evitarem problemes inesperats d'última hora.
- Realitzarem reunions periòdiques amb el director per comprovar l'estat del projecte, per evitar realitzar treball inútil en cas que existeixi confusió sobre els objectius o requeriments del projecte.

2.2 Planificació i fases del projecte

Hem estimat la càrrega de treball del projecte en aproximadament 450 hores, o 13 setmanes, suposant una dedicació de 35 hores setmanals. El projecte comença el setembre, això ens permetrà acabar-lo el desembre, amb aproximadament dos mesos de marge per al torn de lectura de gener.

En aquest apartat explicarem les diferents fases en les quals hem dividit el treball per poder realitzar aquesta estimació.

Planificació inicial

La primera fase es correspon amb l'assignatura GEP, i consisteix en definir completament en un document els paràmetres del projecte, incloent:

- El seu context
- Els seus objectius
- El seu abast
- L'estat de l'art
- La planificació
- La seva gestió econòmica
- La seva sostenibilitat ambiental

Presentació fita inicial

Una vegada definits aquests paràmetres, serà necessari preparar el document i la presentació que s'avaluarà com a fita inicial del projecte.

Anàlisis i disseny

Aquesta fase consisteix en dissenyar l'arquitectura del sistema a realitzar.

Per fer-ho haurem de llegir i entendre la documentació sobre el funcionament dels sistemes d'arxius a Linux, i aplicar els principis adquirits en enginyeria del software per aconseguir un disseny simple i eficaç.

Desenvolupament Programa

Aquesta és la fase principal del projecte, on desenvoluparem el nostre sistema de *application streaming* per contenidors d'aplicacions.

El primer pas serà configurar l'entorn de programació. Després es crearà un joc de proves per al programa i després es desenvoluparà el programa iterativament, seguint la metodologia especificada en un apartat anterior. Al final de cada iteració es provarà el programa amb el joc de proves, es documentarà el progrés i es decidiran els objectius de la següent iteració.

Preparar memòria i presentació

Quan es consideri que ja està acabat, o que no hi ha temps suficient per millorar-lo més, es tancarà el desenvolupament del programa i es passarà a la fase de preparar la memòria final del projecte i la seva presentació.

2.3 Estimació del temps

Hems estimat el temps en hores que comportarà cada fase. Hem intentat ajustar la planificació perquè coincideixi amb les càrregues de treball esperades del TFG i de GEP.

Activitat	Dedicació estimada (hores)	Setmanes aproximades
Planificació inicial	60	2
Presentació fita inicial	20	1
Anàlisi i disseny	50	1
Desenvolupament programa cloudpaging	260	7
Preparar memòria i presentació	60	2
Total	**450**	

Taula 1: estimació de la càrrega de treball

El diagrama de Gantt de la planificació es pot trobar en un annex del projecte.

Cal tenir en compte que el projecte serà realitzat per una sola persona, que s'encarregarà de portar a terme totes les diferents tasques, i a més, cada tasca depèn de l'anterior. Això implica que la realització de les tasques serà totalment seqüencial, i per tant el diagrama de Gantt serà lineal.

2.4 Mètodes de validació i pla d'acció

Al final de cada fase, i a intervals regulars durant la fase de desenvolupament, es realitzarà una reunió amb el director per analitzar l'avanç del projecte i comparar desviacions amb el calendari establert. Aquestes reunions ens serviran per comprovar que la direcció del projecte és correcta, és a dir, que no estem desenvolupant coses que no s'adeqüin a les especificacions.

Durant la fase de programació, també ens servirà per aquest propòsit el joc de proves automàtic que hem mencionat anteriorment.

D'altra banda, la comparació de les desviacions amb el calendari establert ens permetrà establir que la velocitat d'avanç és correcta (desviacions per consum).

Usar una metodologia iterativa durant la fase de desenvolupament ens donarà una gran flexibilitat, ja que ens permetrà abandonar el procés en qualsevol moment i obtenir un resultat que, encara que no sigui tan bo com el que volíem, serà usable. A més, deixarem les funcionalitats de menys prioritat per al final, per poder prescindir d'elles si fa falta.

En ser un projecte que no depèn de recursos externs, les úniques desviacions vindrien de l'autor.

2.5 Eines de desenvolupament

Programació

El programa es desenvoluparà i provarà en un sistema Linux (*Xubuntu 16.04*) utilitzant els llenguatges *C* i *Python*, i eines lliures.

Utilitzarem l'entorn de desenvolupament integrat *NetBeans*, en ser un amb el qual hem treballat en el passat, pel que ja coneixem bé la seva interfície i funcions, en particular amb la seva eina de depuració.

El *NetBeans* proporciona un editor de codi complet i un depurador interactiu per programes C i Python, entre altres llenguatges.

En principi no necessitarem cap altra eina per al codi. Si durant el projecte es considera que fa falta, es podrien afegir altres llibreries, sempre que aquestes siguin de lliure distribució.

Gestió temporal y seguiment

Per fer un seguiment de les hores de treball utilitzarem l'eina gratuïta Harvest[1], que permet cronometrar fàcilment el temps dedicat a cada tasca.

[1] https://www.getharvest.com/

Per fer el seguiment dels errors i les tasques pendents utilitzarem l'eina de seguiment *Asana*[1].

Per editar i guardar les entregues i altres documents relacionats amb el projecte utilitzarem Microsoft Office i Microsoft OneDrive, que guarden automàticament un historial de tots els canvis realitzats.

2.6 Gestió econòmica

Qualsevol projecte requereix recursos. En aquest apartat realitzarem una estimació dels costos del projecte, tant materials com humans, i a partir d'aquí donarem un pressupost del projecte. Proposarem també un mecanisme de control de desviacions.

Costs directes

En aquest apartat, calcularem els costos directes de cada fase del projecte

Recursos humans (RH)

Els principals costs directes en el nostre projecte són els de recursos humans. El projecte es desenvoluparà per una sola persona, pel que haurà d'assumir els rols de gestor de projecte, dissenyador, programador i *tester*. Per calcular els costs de les diverses fases, hem consultat un estudi de remuneració de l'empresa *Page Personnel,* on s'estima el salari de les diferents posicions de treballadors de l'àmbit informàtic [24], i hem agafat el salari estàndard que tindria un treballador *junior* en cada posició:

Rol	Gestor de projecte	Dissenyador	Programador	*Tester*
Cost	30 €/h	13 €/h	9 €/h	11 €/h

Taula 2: Salari estàndard de cada posició

Per tant, el cost de RH de cada fase serà:

Fase	Hores	Rol	Cost
Planificació inicial	60	Gestor de projecte	**1.800€**
Presentació fita inicial	20	Gestor de projecte	**600€**
Anàlisis i disseny	50	Dissenyador	**650€**
Desenvolupament programa	260	80% programador, 20% *tester*	**2.444€**
Preparar memòria i presentació	60	Gestor de projecte	**1.800€**

Taula 3: Cost estimat de cada fase

Amortització de hardware

Per realitzar la planificació, el disseny i la implementació del projecte, necessitarem un sistema hardware apropiat.

Realitzarem una estimació del cost d'amortització del hardware per hora de treball, tenint en compte la seva vida útil aproximada. Suposant que un any té 2000h de treball en total, l'amortització per hora de cada component es calcula amb la fórmula

[1] https://asana.com/

$\frac{\text{preu}}{\text{vida útil (anys)}*2000}$, i l'amortització total en una fase del projecte s'obté multiplicant l'amortització per hora per les hores que utilitzem un dispositiu.

Els productes *hardware* que utilitzarem seran els següents:

Producte	Preu	Vida útil (anys)	Amortització/h
Ordinador ASUS VivoPC K31CD-DE001T	713 €	4	0,0891 €/h
Monitor Dell UltraSharp U2414H	229 €	4	0,0286 €/h
Teclat i Ratolí Microsoft Wireless Desktop 2000	42 €	4	0,005 €/h

Taula 4: costos de productes hardware

Llicències de software

Tots els productes *software* que utilitzem per a aquest projecte són gratuïts. Per tant, no necessitarem cap pressupost de *software*.

Total

Per tant, els costs directes de cada fase seran:

Fase	Hores	Cost RH	Cost amortització *hardware*	Total
Planificació inicial	60	1.800€	7,4€	**1.807€**
Presentació fita inicial	20	600€	2,5€	**603€**
Anàlisis i disseny	50	650€	6,1€	**656€**
Desenvolupament programa	260	2.444€	31,9€	**2.476€**
Preparar memòria i presentació	60	1.800€	7,4€	**1.807€**
Total		**7.294€**	**55,3€**	**7.349€**

Taula 5: costs directes de cada fase

Com es pot veure, els altres costs són negligibles comparats amb els de recursos humans.

Costs indirectes

Altres costs associats amb el projecte són:

- L'electricitat consumida per l'ordinador (suposem 200W durant 450h)
- La impressió de la memòria del projecte
- Per realitzar les proves del programa, utilitzarem servidors virtuals (VPS), per poder executar fàcilment diferents distribucions Linux y poder provar el client en un ordinador separat del servidor

Producte	Preu	Unitats	Total
Electricitat	0,12 €/kWh	90 kWh	11 €
Impressió memòria en color	0,1089 €/pg	160 pgs	17 €
VPS	$0.007/hora	100 h	0,6 €

Taula 6: Costs indirectes del projecte

Contingències i imprevistos

Afegim un 15% extra com a contingència per a imprevistos.

No hi ha riscos que sigui raonable considerar en aquest projecte.

Pressupost final

El pressupost del projecte, per tant, quedaria així:

	Concepte	Valor
Costs directes	Planificació inicial	1.807€
	Presentació fita inicial	603€
	Anàlisi i disseny	656€
	Desenvolupament programa cloudpaging	2.476€
	Preparar memòria y presentació	1.807€
Costs indirectes	Electricitat	11€
	Impressió memòria en color	17€
	VPS	1€
	Contingència para imprevistos (15%)	1.107€
	Total	**8.485€**
	Total amb 21% d'IVA	**10.227€**

Taula 7: pressupost final del projecte

Control de desviacions

Donat que els costs del projecte vénen quasi completament dels recursos humans, el control de la gestió econòmica és equivalent al control temporal, por tant, els mecanismes de control seran els mateixos.

No hi haurà per tant desviacions per preu, només per consum.

Per monitorar-les, mantindrem un informe de gestió econòmica i temporal amb les hores reals de treball. Al final de cada fase, l'actualitzarem amb les hores reals que s'hi han utilitzat. D'aquesta manera es podran comparar els temps estimats amb els reals i ajustar la trajectòria del projecte adequadament.

3 Sostenibilitat

En aquesta secció farem un anàlisi de la sostenibilitat del projecte, seguint el model de la matriu de sostenibilitat definit en la normativa de projectes de la FIB.

Avaluarem 3 tipus d'impacte: ambiental, econòmic i social. Per cada un d'ells, considerarem l'impacte que tindrà el projecte posat en producció (que inclou planificació, desenvolupament i implantació), l'impacte que tindrà el projecte durant la seva vida útil, i finalment, els riscos que puguin incrementar aquests impactes.

3.1 Estudi d'impacte ambiental

Projecte Posat en Producció

El producte realitzat en aquest projecte és un sistema de *software*. Això vol dir que no requereix directament cap recurs material. Tot i això, el *software* evidentment ha de ser executat en ordinadors per tenir alguna utilitat. La pregunta és doncs, si el nostre projecte crearà un increment en el consum dels ordinadors que l'executin.

La fase de realització te un impacte molt clar, que és el consum d'energia de l'ordinador que farem servir durant les 450 hores de treball. Si la potencia mitja d'un ordinador en ús normal està al voltant de 150W, 450 hores ens dona un consum total de 67,5 kWh (243 MJ).

Donat que l'emissió de CO_2 per generació d'electricitat a España es de 308 g CO_2/kWh [25], l'impacte total serà aproximadament 21 kg CO_2. La mitjana d'emissions per persona a la UE és de 8.75 tones per any [26]. Per tant, podem afirmar que l'impacte ambiental de la realització del nostre projecte és més o menys equivalent a un dia de vida normal d'una persona de la unió europea.

Vida útil

Pel que fa a la vida útil del projecte, creiem que també pot tenir un lleuger impacte negatiu en forma de consum d'electricitat, però considerem que és pràcticament impossible fer-ne una estimació.

Recordem que el nostre projecte pretén incrementar la flexibilitat a l'hora de moure contenidors entre ordinadors. Això permetrà un consum més eficient dels recursos, ja que caldrà tenir menys ordinadors "en espera" per si es produeix un increment sobtat de demanda, i els ordinadors passaran també menys temps descarregant programes i més temps executant-los.

Però sabem que existeix un efecte conegut com a *paradoxa de Jevons,* que afirma que, moltes vegades, quan una tecnologia es torna més eficient, el seu consum de recursos total no baixa sinó que puja, a causa de l'increment de demanda que això provoca [27] [28]. Així doncs, és possible que això passi.

També cal mencionar que, com hem explicat a l'apartat d'estat de l'art, ja existeixen actualment sistemes comercials amb la mateixa funcionalitat que el nostre. Això vol dir

que per les empreses de *hosting* que ja els utilitzin, no hi hauria cap diferència en el consum. Per a totes les altres que no els utilitzin degut al seu preu, s'aplicaria el cas explicat.

Riscs

Sobre els riscs, no hem trobat cap escenari que pugui incrementar la petjada ecològica del projecte.

3.2 Estudi d'impacte econòmic

Projecte Posat en Producció

El cost de desenvolupament del projecte s'ha estimat en aquest mateix document en 10.227€, que és poc per un projecte informàtic d'aquesta categoria. La majoria del cost ve dels recursos humans necessaris per fer el disseny i la implementació del projecte, pel que no es pot reduir fàcilment.

Vida útil (viabilitat)

Com hem explicat, els sistemes actuals que realitzen la mateixa funció són comercials. Malauradament, no hem pogut determinar el seu preu, ja que les empreses que els ofereixen no l'anuncien públicament sinó que requereixen que el comprador es posi en contacte amb el seu departament de vendes. Això vol dir, però, que són suficientment cars com per poder permetre's perdre vendes a compradors petits a canvi de poder incrementar el preu als compradors grans.

El nostre sistema en canvi és gratuït, pel que estalviarà diners als usuaris que el facin servir.

No creiem que tingui un cost significant de manteniment durant la seva vida útil, ja que el seu disseny s'ha mantingut intencionalment de baix nivell per maximitzar la seva flexibilitat i mantenir-lo "desacoblat" dels altres elements del sistema. Això vol dir que en principi no caldrà fer-li canvis per adaptar-lo a diferents usos. Tot i això, algunes empreses poden voler fer-li millores o canvis, cosa que comportaria una despesa, però no creiem que això es pugui considerar part de l'impacte econòmic d'aquest projecte sinó que s'hauria de considerar com un projecte totalment nou.

Al ser un sistema de software lliure, és difícil parlar de viabilitat econòmica, ja que el software lliure es finança de manera diferent a la resta de productes. Normalment, es fa amb una combinació de donacions i de contribucions per part d'empreses interessades en disposar d'alguna millora en particular en el producte. Tot i això, si considerem que el seu cost de desenvolupament i manteniment són baixos i que moltes empreses ja paguen per sistemes equivalents (cosa que vol dir que el servei que proporcionen té un valor econòmic alt), creiem que la seva proporció valor-preu és molt bona.

Riscos

Sobre els riscos que puguin fer augmentar el cost del projecte, podem considerar d'una banda el risc que tenen tots els projectes, que és que s'allargui massa la fase de desenvolupament, i d'altra banda podríem considerar un altre risc que és que es descobreixi durant la fase d'implantació que el sistema que alguna de les suposicions que s'han fet en el disseny del sistema no es compleix, i que per tant s'han de fer canvis o millores abans que el sistema es pugui considerar útil a la pràctica. Creiem però que això és molt poc probable, ja que el disseny és simple i realitzarem bastantes proves abans de donar-lo per acabat.

3.3 Social

Projecte Posat en Producció

Tot i que la realització projecte ha tingut a curt termini un impacte molt negatiu en la meva vida personal, creiem que és positiu a llarg termini ja que m'ha ensenyat que si vols fer una feina t'hi has de posar fort per molt poques ganes que en tinguis.

Vida útil

Al proporcionar una alternativa gratuïta als sistemes comercials (tot i que potser amb menys funcionalitat), creiem que el nostre projecte tindrà un impacte social molt positiu en ajudar a reduir la desigualtat, ja que permetrà a usuaris privats i petites empreses que no puguin permetre's les llicencies dels sistemes comercials accedir a la mateixa funcionalitat.

Riscos

No creiem que hi hagi pràcticament cap risc d'impacte negatiu. Podem considerar els treballadors de les empreses que ofereixen aquests productes comercials, però veiem molt poc probable que es produeixi un decrement dels seus ingressos prou alts com perquè perdin la feina.

3.4 Avaluació de sostenibilitat

Tenint en compte l'anàlisi realitzat, hem avaluat cada casella de la matriu de sostenibilitat tal com s'especifica a la guia de l'avaluació de sostenibilitat:

	PPP	Vida útil	Riscos
Ambiental Valoració:	Consum del disseny 9 [0:10]	Petjada ecològica 16 [0:20]	Riscos ambientals -1 [-20:0]
Econòmic Valoració:	Factura 7 [0:10]	Pla de viabilitat 18 [0:20]	Riscos econòmics -4 [-20:0]
Social Valoració:	Impacte personal 5 [0:10]	Impacte social 20 [0:20]	Riscos socials -1 [-20:0]
Valoració total:	21 [0:30]	54 [0:60]	-6 [-60:0]
		69 [-60:90]	

Taula 8: Matriu de sostenibilitat del projecte

Podem concloure doncs que el projecte és molt sostenible.

4 Els sistemes d'arxius en Linux

En aquest apartat explicarem el funcionament dels sistemes d'arxius en Linux, per tal de donar el context en el qual es situarà el nostre sistema i poder entendre el seu disseny.

4.1 El nucli Linux

Linux és un nucli de sistema operatiu, creat el 1991 per Linus Torvalds i distribuït amb llicencia GPL, que permet la seva còpia i modificació sempre que aquestes modificacions també es distribueixin amb llicencia GPL.

Va ser originalment creat com a substitut del sistema operatiu *Minix*, que alhora va ser publicat el 1987 com a clon del sistema operatiu *Unix V7*, publicat el 1979.

És per tant un sistema operatiu *Unix-like*, és a dir que implementa la mateixa interfície que implementava el sistema operatiu *Unix*. Això no vol dir que sigui simplement una còpia de *Unix*, ja que ha anat afegint a més interfícies i funcionalitats pròpies més modernes.

En el nostre projecte desenvoluparem només per Linux, és a dir, que no ens preocuparem de separar la funcionalitat comuna de *Unix* i la específica de Linux.

Com tot sistema operatiu, Linux executa processos en *user mode*, cosa que vol dir que no poden accedir directament al hardware ni a cap part de memòria que no sigui la seva. Els processos es comuniquen amb l'exterior mitjançant trucades al sistema (*system calls*). Una trucada al sistema és una instrucció especial que permet saltar al codi del sistema operatiu, normalment per demanar-li a aquest que faci alguna operació d'entrada i sortida i retorni.

4.2 Operacions d'arxiu en Linux

Una de les operacions més importants que qualsevol sistema operatiu ha d'oferir als seus programes és la de llegir i escriure dades en memòria persistent, és a dir, llegir i escriure arxius.

Com els altres *Unix*, Linux té un sistema d'arxius jeràrquic, amb un sol directori arrel, conegut com "/", on cada directori pot contenir més directoris i arxius. Un arxiu doncs es pot identificar pel seu camí complet (el seu nom i el nom dels directoris que el contenen, separats pel caràcter "/").

Les sis operacions més importants que cal conèixer del sistema d'arxius són:

- `open`/`close`: obren i tanquen arxius (explicat a continuació)
- `read`/`write`: llegeixen i escriuen bytes en una posició d'un arxiu obert
- `stat`: obté les metadades d'un arxiu. Inclou informació com mida, permisos, data de creació i modificació
- `getdents`: obté la llista d'arxius en un directori

Abans de poder fer la majoria d'operacions amb un arxiu, el procés l'ha de "obrir" explícitament amb la trucada *open*. El procés rep un descriptor d'arxiu del sistema (que és simplement un número identificador arbitrari) i l'utilitza en totes les operacions posteriors per identificar l'arxiu sobre el que vol operar. Quan acaba, el procés l'ha de tancar amb la trucada *close*.

Aquest disseny te diversos avantatges: simplifica el codi del sistema, redueix el cost de les operacions més freqüents, i permet al sistema saber quins arxius estan en ús per protegir-los de ser esborrats.

4.3 Arxius especials

A més dels arxius i directoris "normals", Linux té diversos altres tipus d'objectes que apareixen en el sistema d'arxius. Els més importants són:

1. Enllaços simbòlics: són objectes que contenen una referència a un altre arxiu o directori. Quan un programa els intenta obrir, el sistema segueix la referència i obre l'arxiu a què apunta, a menys que el programa especifiqui explícitament el contrari.

   La referència s'emmagatzema de forma textual. Això vol dir que el destí pot ser un destí relatiu o absolut, pot no existir, o fins i tot pot crear-se per accident un bucle d'enllaços simbòlics.

2. Enllaços durs ("*hard links*"): dos o més arxius que apunten als mateixos blocs de dades al disc. No són tècnicament diferents als arxius normals des del punt de vista de l'aplicació, però són importants perquè si es modifica un dels arxius, els canvis es veuen immediatament als altres.

3. Canonades amb nom ("*named pipes*"), sòcols ("*sockets*"), i arxius de dispositiu ("*device files*"): no són realment arxius del disc, sinó que s'utilitzen per comunicar-se amb altres processos o amb dispositius hardware del sistema. Aquests objectes es poden llegir i escriure amb les mateixes crides que els arxius. Aquest disseny prové de Unix, i es coneix com la filosofia "tot és un arxiu".

4.4 Mòduls i muntatge

Per poder adaptar-se a diferents entorns i oferir funcionalitats noves, la implementació dels sistemes d'arxius a Linux és totalment modular. És a dir, el nucli Linux delega totes les operacions a diferents mòduls, que són els que s'encarreguen d'escriure i llegir les dades al suport físic real.

Això permet, per exemple, tenir compatibilitat amb molts formats de sistema d'arxius en disc diferents, incloent els que utilitzen d'altres sistemes operatius, com FAT32 o NTFS, o fins i tot utilitzar altres tipus de sistemes d'arxius que no estiguin guardats en disc, com sistemes en xarxa o sistemes generats dinàmicament.

Per poder utilitzar un d’aquests mòduls s’ha de “muntar” en un directori. Això fa que el sistema dirigeixi totes las operacions dins d’aquell directori al mòdul especificat.

Així, si tenim un directori */media/DiscUSB1/*, i tenim connectat un disc USB en format FAT32, podem muntar el disc en el directori utilitzant el mòdul *vfat*. Aleshores, si un procés intenta llegir l’arxiu */media/DiscUSB1/arxiu1*, el nucli Linux demanarà *arxiu1* al mòdul *vfat*, i aquest mòdul el buscarà en el disc extern. L’efecte pràctic d’això és que veurem el contingut del disc USB en el directori */media/DiscUSB1/*.

Això dona més flexibilitat que altres sistemes operatius, com per exemple Windows, on cada disc és una arrel del seu propi arbre d’arxius. Per exemple, l’usuari pot moure el contingut de qualsevol carpeta a un disc propi, muntar aquell disc en la mateixa carpeta, i el sistema continuarà funcionant igual que abans.

4.5 La interfície Virtual File System

La interfície interna que el nucli Linux fa servir per comunicar-se amb els mòduls que implementen sistemes d’arxius es coneix com a *Virtual File System*, o *VFS*.

VFS defineix un conjunt de funcions i estructures de dades, molt similars (tot i que no idèntiques) a les que el nucli Linux exposa als processos. El codi intern de Linux rep aquestes trucades i les dirigeix al mòdul corresponent a través d’aquestes funcions.

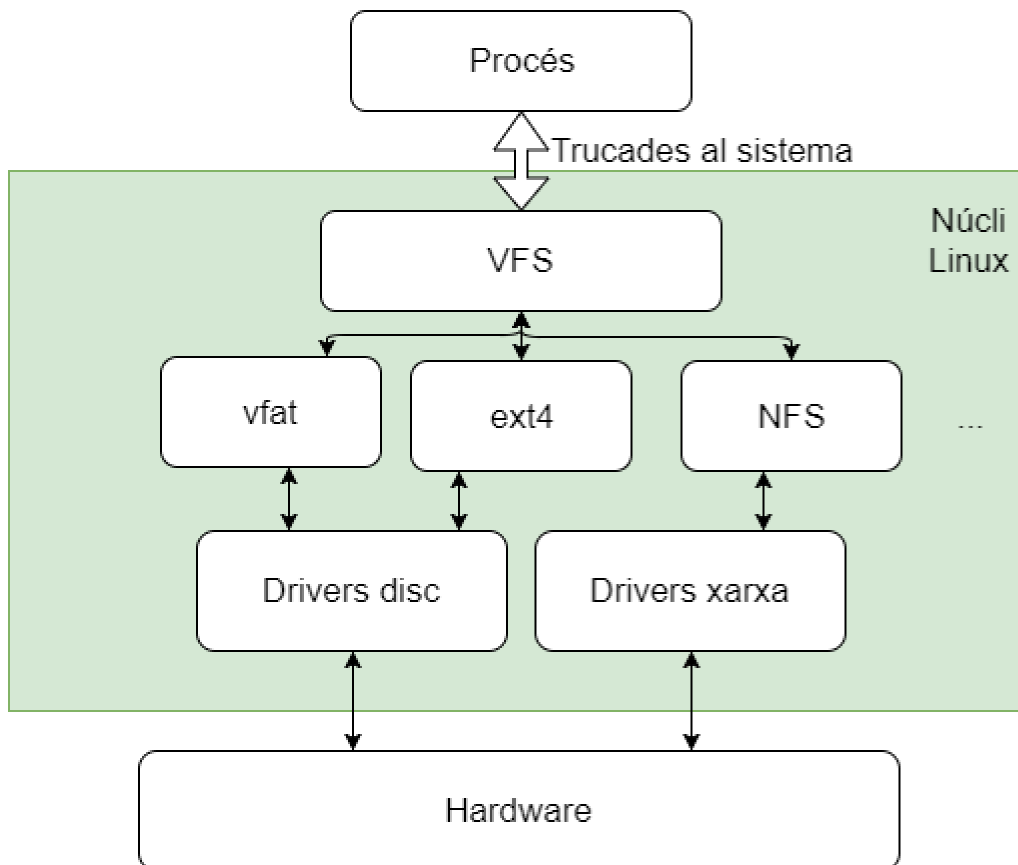

Figura 1: diagrama del funcionament dels sistemes d’arxius VFS

4.6 FUSE (Filesystem in Userspace)

FUSE significa "*Filesystem in Userspace*". FUSE es un mòdul del nucli Linux que, com el nombre suggereix, permet implementar sistemes d'arxius en mode d'usuari, cosa que permet major flexibilitat. Està inclòs en Linux des de la versió 2.6.14.

FUSE implementa la interfície VFS, actuant com a sistema d'arxius de cara al nucli Linux, però reenvia totes les peticions a un procés d'usuari mitjançant un *socket*.

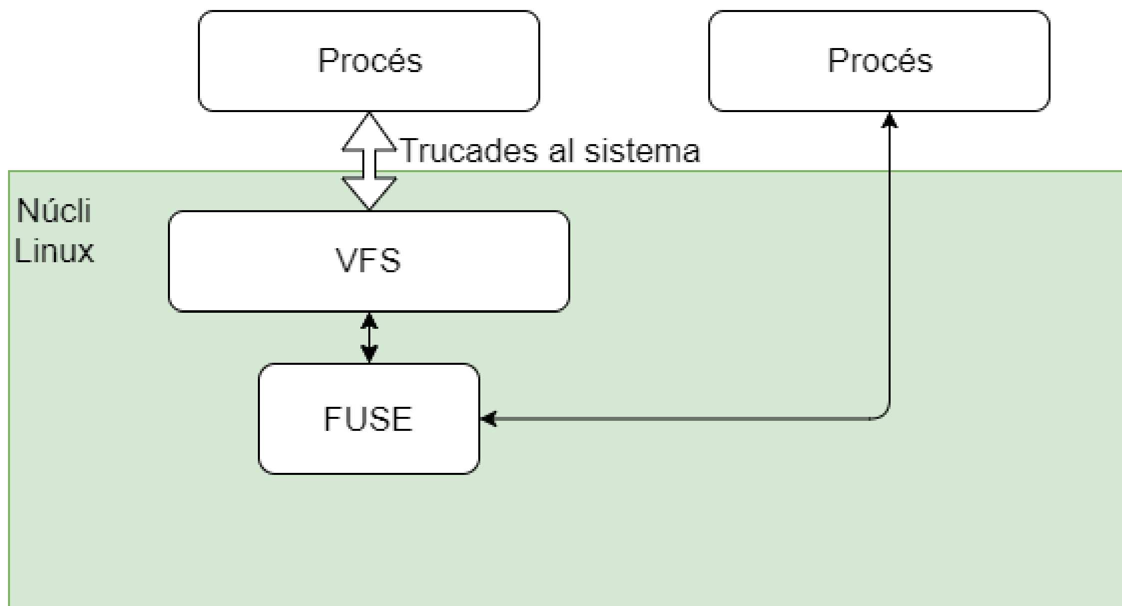

Figura 2: diagrama del funcionament del mòdul FUSE

El principal desavantatge de FUSE respecte a implementar directament els sistemes d'arxius amb VFS és un lleuger decrement de velocitat, en haver de reenviar totes les peticions. En la majoria de sistemes, però, el coll d'ampolla en els accessos d'arxius està en l'accés al dispositiu físic o de xarxa, pel que aquest temps extra no és important.

La seva senzillesa l'ha convertit en un sistema popular per implementar gran quantitat de sistemes d'arxius que ofereixen tota mena de funcions variades. Podem trobar, per exemple, sistemes que ofereixen accés a serveis en xarxa, que converteixen formats d'arxius de manera transparent, que ofereixen accés a formats contenidors (com zip), o que implementen nous conceptes com sistemes d'arxius no jeràrquics [29].

Fins i tot el sistema *NTFS-3G*, la principal implementació oberta del sistema d'arxius NTFS que utilitza Windows, utilitza FUSE [30].

5 Disseny del sistema

En aquest apartat descriurem el disseny del nostre sistema. Primer explicarem l'enfocament general que hem fet servir per elaborar-lo.

5.1 Enfocament del disseny: l'algoritme de cloudpaging

El contenidor que volem executar està compost de diversos arxius. El sistema d'execució de contenidors s'encarrega de llegir aquests arxius, interpretar-los i executar-los d'alguna manera.

Per aconseguir doncs que aquest sistema pugui executar aquests contenidors al mateix temps que es descarreguen, sense haver de fer-li cap canvi intern, haurem de crear un programa que munti un sistema d'arxius en un directori del sistema, en el qual "oferirà" els arxius del contenidor, i s'encarregui de transferir aquests arxius en temps real a mesura que facin falta. Així, qualsevol altre procés podrà accedir als arxius en aquell directori igual que si estiguessin guardats en un disc dur local, quan en realitat s'estan obtenint d'un servidor. Aleshores simplement hem de dir-li al sistema de contenidors que els executi des d'allà.

Per transferir i emmagatzemar aquestes dades, dividirem cada arxiu en blocs de la mateixa mida, i transferirem cada bloc en el moment en què es llegeixi per primer cop. Aquests blocs s'emmagatzemaran en un directori *cache* del sistema, per no haver-los de re-descarregar quan tornin a fer falta.

Aquest algoritme es coneix com a *cloudpaging* [31], per la seva similitud al sistema de paginació de memòria (*paging*) que els sistema operatius fan servir per carregar i descarregar blocs de memòria al disc.

5.2 Disseny

Per tant, doncs, el nostre sistema funcionarà de la següent manera:

Crearem un sistema servidor-client, que s'executi en dos ordinadors. El servidor serà l'ordinador que té inicialment els arxius, mentre que el client serà l'ordinador que volem que executi el contenidor.

La part de client ha de:

1. Establir connexió amb el servidor
2. Muntar un sistema d'arxius en un directori del sistema
3. En rebre qualsevol operació de sistema d'arxius, presentar la mateixa estructura d'arxius que té el servidor en aquell directori, amb les mateixes metadades
4. En rebre una operació de lectura sobre un arxiu:
    1. Comprovar si els blocs necessaris per realitzar l'operació ja s'han obtingut
    2. Si no, obtenir els que faltin del servidor i guardar-los
    3. Entregar-los al programa

La part de servidor simplement ha de rebre les peticions del client i transferir-li les dades quan faci falta.

5.3 Estructures de dades

Per servir els arxius de manera eficient, el sistema haurà de tenir-los en memòria.

Això inclou tres elements principals: l'estructura dels arxius, els seus atributs (metadades), i les seves dades.

A nivell lògic, aquesta informació es pot representar amb el següent diagrama UML:

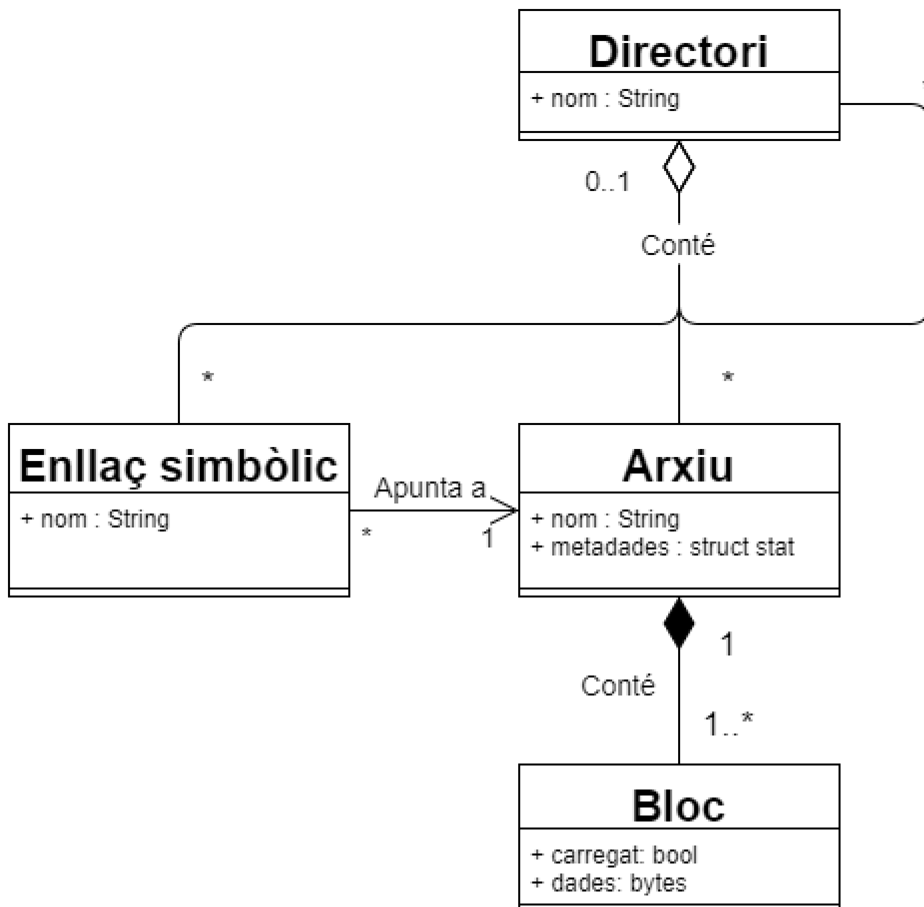

Figura 3: estructures de dades del sistema

Podem veure una estructura jeràrquica de directoris, que haurà de replicar l'original dels arxius que vulguem servir, un conjunt d'arxius amb el seu nom i atributs (el tipus de dades *struct stat* és el que representa en Linux els atributs d'un arxiu), i una llista de blocs de dades, que composen l'arxiu.

Per cada bloc, haurem de guardar dues coses: informació sobre el seu estat (descarregat o no descarregat) i les dades del bloc. Com que les dades dels blocs seran molt grans, no les podrem guardar en la memòria RAM, sinó que les haurem de guardar en el disc. La informació del seu estat, en canvi, serà un sol bit, o sigui que la guardarem en memòria.

6 Sistema d'anàlisi d'us d'arxius

Hem decidit crear, com a complement al sistema principal de *cloudpaging*, un segon sistema que pugui interceptar i analitzar totes les operacions de lectura i escriptura que efectuï qualsevol altre programa.

És a dir, quan un programa determinat efectuï una crida *read, open* o *write* a qualsevol arxiu, amb uns certs paràmetres, el nostre programa ha de rebre aquesta informació d'alguna manera. Això ens permetrà recollir aquestes dades i calcular a partir d'elles qualsevol mètrica o estadística que vulguem.

El motiu pel qual volem crear aquest sistema és doble: d'una banda, ens servirà en aquest treball per justificar l'eficàcia del sistema de cloudpaging. D'altra banda, el sistema podrà ser utilitzat en un futur per optimitzar l'algoritme de cloudpaging. Per exemple, afegint una funció de "descàrrega predictiva", que descarregui els blocs amb més probabilitat de fer falta en el futur proper.

El sistema estarà doncs dissenyat com a "infraestructura" que podrà ser adaptada a obtenir qualsevol estadística segons facin falta.

6.1 Disseny

Linux inclou una funció anomenada *ptrace* que permet interceptar les trucades a sistema de qualsevol procés. Així doncs, podríem simplement fer-la servir per interceptar les trucades *open* i *read* del procés que ens interessés.

Però donat que haurem d'implementar un sistema d'arxius per l'altre sistema, hem decidit aprofitar part de la feina i fer servir el mateix enfocament aquí.

Per tant, el que farem serà crear un sistema d'arxius que "dupliqui" el sistema d'arxius del sistema, redirigint les peticions que rebi. Aleshores executarem el programa dins d'aquest sistema d'arxius mirall, mitjançant la trucada al sistema *chroot* que permet fer això.

Així aconseguirem un programa que rebi totes les operacions de lectura i escriptura que ens interessen. Aleshores serà trivial afegir codi per apuntar-se les operacions en una estructura de dades apropiada i calcular qualsevol mètrica que ens interessi sobre aquestes.

7 Desenvolupament

7.1 Part comuna: sistemes d'arxius

Hem de desenvolupar doncs dos programes que implementin sistemes d'arxius. En aquest apartat descriurem els detalls de com implementarem aquesta part comuna.

Llenguatge C

Com ja hem dit en un apartat anterior, farem aquesta part del programa en llenguatge C. Fem servir C principalment perquè és el llenguatge estàndard que fan servir totes les llibreries i eines de Linux, i ens facilita la interacció amb elles. A més, és un llenguatge amb el que ja tenim experiència programant, i és compilat i baix nivell, cosa que vol dir que s'executa més ràpidament.

Mòdul FUSE

Hem decidit fer el programa utilitzant el mòdul FUSE en comptes d'interactuar directament amb la interfície VFS.

Com hem explicat anteriorment, el mòdul FUSE facilita significativament el desenvolupament comparat amb VFS, però fa que el sistema sigui lleugerament menys eficient. En el nostre cas ja hem decidit que l'eficiència del sistema és una prioritat baixa, per tant aquest petit sacrifici no ens importa. A més, és molt probable que el retard afegit per FUSE sigui negligible comparat amb el retard inevitable de la petició de xarxa.

Llibreria libfuse

Libfuse[1] és la llibreria associada a FUSE, mantinguda per els mateixos autors de FUSE, per facilitar el seu ús. Està escrita en C, i es distribueix sota la llicència LGPL.

Libfuse s'encarrega de comunicar-se amb el *socket* de FUSE, i ofereix una interfície de trucades de retorn ("*callbacks*") al programador.

Interfícies

La llibreria ofereix dos interfícies: una d'alt nivell i una de baix nivell. La principal diferència és que la de baix nivell requereix que el programador gestioni la correspondència entre noms d'arxiu i *inodes* (una estructura de memòria que identifica un arxiu), i que recordi i respongui explícitament a cada petició, mentre que amb la d'alt nivell se n'encarrega la pròpia llibreria.

Utilitzarem la d'alt nivell, ja que la de baix nivell comporta molt més treball sense proporcionar, en el nostre cas, cap avantatge.

[1] https://github.com/libfuse/libfuse

Versions
En desembre de 2016, libfuse va publicar la seva versió 3.0. Aquesta versió és important perquè trenca la compatibilitat amb les versions anteriors. Així doncs, libfuse es troba ara dividit en dues branques paral·leles no compatibles: libfuse2 y libfuse3.

Per decidir quina versió utilitzar, hem buscat en els dipòsits de programari de les principals distribucions de Linux per veure quina és la última versió de libfuse que inclouen. Hem buscat en dues versions de cada distribució (l'última i la penúltima):

Distribució	Última versió de libfuse disponible en escriure aquest document
Debian 8	2.9.3
Debian 9	2.9.7
Ubuntu 16.04LTS	2.9.4
Ubuntu 17.10	2.9.7
CentOS 6	2.8.3
CentOS 7	2.9.2
openSUSE 42.3	2.9.3
openSUSE Tumbleweed	2.9.7
Fedora 26	2.9.7
Fedora 27	3.1.1

Taula 9: versions de libfuse disponibles

Veiem que només una distribució té ja libfuse3, i només CentOS 6 té una versió inferior a 2.9.

Per tant, si desenvolupéssim el codi per la versió 3, hauríem d'incloure també la llibreria en distribuir el programa, cosa que comportaria més feina. En canvi, si desenvolupem per qualsevol versió 2.9.x, hauria de funcionar directament en qualsevol distribució Linux (excepte possiblement CentOS 6), ja que les diferències en versions menors no haurien de causar problemes de compatibilitat.

Funcionament
Libfuse defineix un conjunt d'operacions que es corresponen, tot i que no al 100%, amb les operacions a fitxers de Linux: obrir arxiu, llegir arxiu, escriure arxiu, obtenir el contingut d'un directori, etc.

El programa ha d'implementar una funció C per cada operació i registrar-la amb libfuse. Libfuse després s'encarrega de cridar-la amb els paràmetres de cada operació i de reenviar el que retorni la funció la funció al sistema.

Totes les operacions són opcionals. Si el programa no implementa una operació, libfuse en alguns casos proporciona una implementació per defecte, si no simplement retorna un error al programa que l'hagi sol·licitat.

No ens hem de preocupar de gestionar els enllaços simbòlics, ja que el mòdul FUSE dirigeix la crida directament al seu arxiu de destí.

7.2 Sistema d'anàlisi d'us d'arxius

En aquest apartat descriurem els detalls de la implementació del sistema d'anàlisi d'us d'arxius.

Base

Part del codi d'exemple que ve inclòs amb la llibreria libfuse és un programa que redirigeix les peticions a l'arrel del sistema d'arxius, creant un "mirall" del sistema en el directori on es munti. Per exemple, si el muntem en el directori */tmp/dir1*, veurem tots els directoris de l'arrel (*usr/*, *etc/*, *home/*, *proc/*, *mount/, bin/*…) dins de */tmp/dir1*.

Això és molt convenient per nosaltres ja que és exactament el programa que necessitem. A més, el codi està organitzat en una funció per cada operació de fitxers, per tant és fàcil afegir codi a les que volem. Així doncs, el farem servir com a base.

L'únic desavantatge que té fer això és que el codi d'exemple té llicència GPL, per tant estem obligats a mantenir aquesta llicència al crear un producte derivat d'ell. Això ens obliga a distribuir el codi juntament amb l'executable. Però no és cap problema per nosaltres ja que no teníem cap intenció de "tancar" el codi.

Codi Python

Calcular estadístiques en C pot ser molt complicat, ja que és un llenguatge de baix nivell no orientat a càlculs matemàtics ni a manejar estructures de dades complexes. Per això hem decidit fer aquesta part en Python.

Python és un llenguatge de programació d'alt nivell, interpretat, i orientat a objectes. L'intèrpret està preinstal·lat en totes les distribucions de Linux, pel que no suposa cap dependència addicional. El principal desavantatge de Python respecte C és que és significativament més lent, però en aquest cas la velocitat no ens importa.

Una altra opció seria exportar totes les operacions a un arxiu, que després podríem importar a un sistema d'anàlisi estadístic com *R* o *Matlab*, però preferim Python perquè hi tenim més experiència.

Un altre avantatge de Python és que disposa de gran varietat de llibreries senzilles per crear gràfics, interfícies gràfiques, o qualsevol altra funcionalitat que puguem voler afegir en un futur. Per exemple, podríem crear una interfície que mostrés un gràfic interactiu amb les dades recollides en temps real.

Aquest codi Python serà doncs el que manipularem per guardar i calcular les estadístiques que ens interessin. Aquestes dades calculades es podran exportar a un arxiu *.csv*, que es pot obrir fàcilment amb un programa com LibreOffice per crear-ne gràfics.

Comunicació entre les dues parts

Per tant, haurem de crear una interfície entre el programa escrit en C i el codi Python.

Per fer això, el que farem serà afegir funcions *printf* en els punts del que ens interessin, que imprimeixin la informació que vulguem per la sortida estàndard del procés, en un format específic.

Aleshores, el programa Python executarà l'altre programa i rebrà aquests missatges a través d'una canonada (*pipe*). Així podrem obtenir les operacions realitzades en els arxius en temps real.

Execució del programa a analitzar

Un cop tenim una versió "mirall" de tot el sistema d'arxius en un directori, hem d'aconseguir executar el programa que volem "dins" d'aquest directori per tal que les peticions que efectuï passin per el nostre programa.

Linux inclou una funció anomenada *chroot* ("change root"), que modifica un procés per definir un directori qualsevol com a nou directori arrel del procés. És a dir, exactament el que volem.

Aquesta funció, però, té un inconvenient important: només es pot cridar en un procés que s'executi com a superusuari ("*root*"). Nosaltres necessitem executar els programes com al mateix usuari que utilitzem normalment, per evitar modificar el seu comportament (a més, executar programes com a superusuari sempre comporta un risc de seguretat que ens interessa minimitzar).

Això vol dir que necessitaríem crear un nou procés elevat amb *sudo*, fer *chroot* dins d'aquest procés, tornar a baixar a usuari normal i finalment cridar el programa que ens interessa.

En comptes d'això, hem decidit utilitzar l'eina *fakechroot*. *Fakechroot* és un programa que simula els efectes de la crida *chroot* sense necessitar permisos de superusuari, interceptant i modificant les trucades del sistema del procés que volem. És una eina molt utilitzada, creada precisament per evitar aquest problema, pel que estarà disponible a totes les distribucions.

7.3 Sistema de cloudpaging

En aquest apartat descriurem els detalls de la implementació del sistema de cloudpaging per contenidors d'aplicacions.

En aquest programa no hem de passar peticions al sistema d'arxius, sinó que les hem de gestionar de manera més complicada dins del programa. Per tant, no podem aprofitar codi de l'altre programa, però si part dels coneixements adquirits.

Suposicions

Al desenvolupar el programa, hem suposat que:

1. Els arxius del servidor són totalment estàtics, i no canvien en cap moment mentre el programa està iniciat.

2. Els contenidors només estan compostos de carpetes, arxius, i enllaços simbòlics (no hi ha altres tipus d'arxiu com canonades o arxius de dispositiu)
3. Els contenidors tenen entre 1 y 1000 arxius (normalment més a prop de 1)

Comunicació amb el servidor

La part més important del sistema és transferir els blocs de servidor a client quan toca.

Una possibilitat era escriure el codi nosaltres mateixos. Aquest codi s'hauria d'encarregar d'establir la connexió entre el servidor i el client, definir un protocol de xarxa, enviar les peticions de blocs d'arxius al servidor i enviar les dades sol·licitades al client. Això, però, seria mala pràctica, ja que estaríem reinventant una solució per a un problema que ja ha estat "resolt" moltes vegades. Hem decidit doncs reutilitzar una solució existent.

Hem buscat sistemes que permetin a un ordinador Linux accedir a fragments arbitraris (accessos aleatoris o seqüencials) d'un arxiu en un altre ordinador Linux, de manera eficient. Les solucions que hem considerat vàlides són:

- *Network File System (NFS)*: sistema client-servidor dissenyat per (com el nom indica) muntar sistemes d'arxius remots, molt popular en Linux.
- *Server Message Block (SMB)*: també sovint anomenat *Common Internet File System (CIFS)*, tot i que en realitat són protocols diferents. Sistema client-servidor, també dissenyat per muntar sistemes d'arxius remots. Desenvolupat per Microsoft i usat nativament per Windows per compartir arxius, però és un protocol obert i amb implementacions per Linux.
- *SSH File Transfer Protocol (SFTP)*: extensió de SSH per transferir arxius. El servidor està inclòs en el servidor SSH estàndard, cosa que facilita el seu ús en servidors Linux.

Les solucions que hem identificat però descartat són:

- *Rsync*: programa dissenyat per sincronitzar arxius grans tant entre ordinadors como dins d'un mateix ordinador. Computa un "*rolling checksum*" de l'arxiu per identificar les parts diferents i enviar-les. Per tant, no serveix per accessos aleatoris.
- *File Transfer Protocol (FTP)*: protocol dissenyat per transferir arxius. Malgrat estar dissenyat per al propòsit que volem, és un protocol molt antic, considerablement més insegur i ineficient que els altres protocols més moderns. Per tant hem decidit no utilitzar-lo.
- *Secure Copy* (SCP): programa que còpia arxius sencers sobre una connexió SSH estàndard, similar a SFTP però molt més simple. No ens serveix perquè no permet llegir trossos parcials
- *HTTP*: es pot utilitzar per transferir parcialment arxius, però no és el seu propòsit principal.

Veiem que NFS, SMB i SFTP, tot i ser internament protocols completament diferents, són bastant similars respecte al seu ús i les seves funcionalitats, i són igualment viables i eficients per el nostre ús.

Hem triat doncs NFS com a protocol principal perquè creiem que és el més popular. Tant el client com el servidor estan disponible en tots els sistemes Linux i en la majoria de no Linux.

Al estar dissenyat com a sistema d'arxius, per transferir un arxiu amb NFS, el nostre programa ha de muntar el servidor en una carpeta local i llegir d'allà.

Això vol dir que, un cop muntat el directori, el nostre programa transferirà arxius simplement llegint-los d'un directori i escrivint-los a un altre. Això té l'avantatge que podem substituir NFS amb qualsevol altre protocol que implementi la mateixa interfície, i el programa seguirà funcionant.

Per tant doncs, hem escrit el programa de manera que l'usuari li pugui passar o bé un nom de servidor NFS per establir automàticament la connexió, o bé un directori on ja estigui muntat un sistema d'arxius en xarxa.

Estructures de dades

Com hem explicat en un apartat anterior, hem de guardar l'estructura dels arxius que volem servir, les seves metadades, les seves dades, i l'estat de cada bloc. Per guardar els blocs de dades, necessitarem per força un directori *cache* al disc.

El directori cache

Hem decidit guardar els arxius en el directori cache amb els mateixos noms i la mateixa estructura que al servidor.

Per guardar les dades, aprofitarem la funcionalitat de *sparse files* que implementa Linux. Aquesta funcionalitat permet crear arxius d'una mida arbitrària, en els que no s'ocupa físicament l'espai en el disc fins que no s'escriu en una part per primer cop.

És a dir, podem crear instantàniament un arxiu de 10GB, escriure 1MB de dades en qualsevol posició de l'arxiu, i només s'ocuparà aproximadament 1MB d'espai en el disc. Les parts no inicialitzades dels arxius s'anomenen "forats", i qualsevol programa que les llegeixi veurà només bytes "00".

Són per tant ideals per el nostre sistema, ja que podem inicialitzar-los a la mida original dels arxius del servidor, i anar emplenant els blocs de dades a la posició que els correspon. Això a més ens permet guardar i obtenir l'estructura dels arxius i directoris.

Com que assumim que els arxius del servidor no canvien durant l'execució, llegirem i inicialitzarem completament aquesta estructura d'arxius a l'inici del programa. Això ens simplifica molt el codi.

Informació dels arxius

Les metadades dels arxius (*struct stat*) i l'estat dels blocs si que els guardarem explícitament en memòria.

Necessitarem consultar aquestes dades en cada petició d'arxiu. Per tant, les hem de guardar en una estructura que permeti trobar-les a partir del *path* de manera molt eficient.

Una estructura que permet trobar un valor a partir d'una cadena de text de manera eficient es coneix com *array associatiu*. La seva implementació més comuna és la taula hash ("*hash table*" o "*hash map*").

Així doncs definirem un tipus de dades *struct fileinfo* que contindrà la informació que volem per cada arxiu, i una taula hash que associï el camí de cada arxiu amb la seva informació.

Implementació de la taula hash
Com que implementar una taula hash eficient és complicat, hem buscat implementacions existents per reutilitzar-les. Hem considerat les següents opcions:

1. La implementació inclosa en la llibreria C de Linux (funcions *hcreate* y *hsearch*) [32]. Té l'avantatge evident que la podem fer servir directament sense haver d'instal·lar res.

   Malauradament, la llibreria està seriosament limitada. Només permet tenir una sola taula en memòria, no permet expandir-la després de crear-la, iterar per els elements guardats, ni esborrar-los. Això vol dir que si es necessita en algun moment poder iterar per tots els elements, s'hauran de guardar en una segona llista a part, i si es vol expandir la taula per inserir més elements, s'haurà d'esborrar-la completament i tornar-la a recrear amb una mida més gran.

2. La implementació inclosa en la llibreria STL de C++. Com que C++ és un llenguatge compilat similar a C, però amb una llibreria estàndard molt més extensa, i que també està inclosa per defecte en gairebé tots els sistemes Linux, podríem teòricament fer "préstec" d'alguna de les seves funcions des del codi C. El desavantatge és que el codi per trucar funcions de C++ des de C pot ser complicat.

3. Fer servir una llibreria externa. Aquesta seria l'opció "normal" que es faria servir en la majoria de projectes de software, però en el nostre cas estàvem intentant mantenir les dependències al mínim. Buscant en els dipòsits de software hem identificat sis llibreries que podrien servir: *glib*, *libjudy*, *libdhash1*, *libhashkit2*, *uthash-dev*, *librhash0*

Hem decidit finalment fer servir les funcions de la llibreria C, tot i les seves limitacions. De moment establirem un límit de 1000 arxius. Si en un futur fes falta, podríem ampliar aquest límit, implementar una funció per ampliar la taula o substituir aquestes funcions per una llibreria externa.

Joc de proves

Com hem dit a l'apartat de metodologia, per comprovar el funcionament del sistema durant el desenvolupament hem creat també un joc de proves automàtic.

El joc de proves consisteix en un script en *Python* que pot crear un conjunt de directoris i arxius aleatoris en qualsevol carpeta, i comprovar que aquests mateixos directoris i arxius aleatoris es poden llegir en qualsevol altra.

Per aconseguir això, aprofitem que *Python* inclou un generador de números pseudo-aleatoris determinista. Això vol dir que pot generar una quantitat de dades molt gran, que serà el mateix a cada execució del programa. Per tant, podem generar milers d'arxius aleatoris en un directori amb poques línies de codi, i després les mateixes línies de codi poden comprovar que aquests arxius son els mateixos, sense haver de guardar-ne una còpia extra.

Això és molt útil per comprovar el nostre sistema de cloudpaging, ja que podem generar els arxius en un directori del servidor, executar el sistema per veure'ls des del client, i executar el joc de proves per comprovar que es llegeixen completament sense errors.

Concretament, el nostre joc de proves crea:

- Un directori amb 5000 arxius, per comprovar que no hi ha cap problema amb directoris molt grans
- Un altre directori amb 3 directoris aleatoris, cada un amb 3 directoris més, i així fins a 6 nivells. És a dir, un arbre amb 3^6 =729 directoris.
- Un arxiu de 1GB

Seguretat

En qualsevol sistema de software que interactuï amb altres sistemes, s'han de tenir en compte les implicacions de seguretat i els possibles "forats" que aquest pugui crear.

En el nostre sistema, hem evitat la majoria de potencials problemes de seguretat delegant al sistema NFS l'autenticació i la transferència segura de dades entre servidor i client. Assumim que l'administrador de sistemes controla tant el servidor com el client, i ha configurat el servidor d'arxius correctament.

El vector d'atac restant, doncs, seria que un usuari creés un contenidor que, al ser transferit amb el nostre sistema, exploti algun error que pugui donar-li control del sistema.

Malauradament, no hi ha cap manera de prevenir totalment vulnerabilitats en codi que gestioni dades insegures. Al ser un sistema acadèmic, tampoc hem dedicat temps explícitament a fer anàlisis de seguretat del nostre codi, però donat que només copia blocs d'arxius i no interpreta mai la informació que contenen aquests, ens sembla poc probable que tingui algun problema d'aquest tipus.

8 Resultats

8.1 Sistema de cloudpaging

A continuació analitzarem els resultats del sistema de *cloudpaging* desenvolupat, en quina mesura compleix els diversos objectius que ens havíem proposat, i el seu funcionament en general.

Presentarem també les proves que hem fet per comprovar el funcionament del nostre sistema i farem un anàlisi dels seus resultats.

El més important a destacar és el fet que el sistema funciona correctament, i ofereix tota la funcionalitat que desitjàvem.

S'han pogut assolir amb èxit, doncs, tots els objectius que s'havien definit en el primer capítol del document.

Joc de proves

El sistema passa perfectament el joc de proves que hem descrit anteriorment. L'hem provat tant localment, com obtenint les dades d'un servidor extern.

8.2 Sistema d'anàlisi d'us d'arxius

El sistema d'anàlisi d'us d'arxius també funciona correctament i proporciona tota la funcionalitat especificada, amb una sola limitació que descriurem a continuació.

L'únic obstacle que hem trobat durant el desenvolupament del programa ha estat que el mòdul *Python* del programa *NetBeans*, que fèiem servir per al desenvolupament de la resta del programa C, no funcionava. Com que necessitàvem un depurador interactiu, hem hagut de buscar un programa alternatiu (hem utilitzat *Eric Python IDE*).

L'hem provat obtenint diverses mètriques en diversos programes d'escriptori. Hem triat tres programes d'escriptori per fer les proves: el navegador web *Firefox*, el paquet d'oficina *LibreOffice*, i l'entorn de desenvolupament *NetBeans*. Hem triat aquests programes perquè són grans, estan compostos de múltiples sistemes interns diferents, i creiem per tant que són bons jocs de proves.

El programa exporta les dades a un arxiu *.csv*. Això permet importar-les a un programa com *LibreOffice* per poder fer-ne gràfics.

Limitació de funcionalitat de *sockets*

En les proves inicials hem descobert que algunes funcionalitats dels programes executats dins del sistema no funcionen correctament. Concretament, hem vist que els programes no emeten àudio, ni poden mostrar gràfics 3D.

Investigant aquest problema, hem conclòs que això és degut a que els "*sockets*" locals del nostre sistema d'arxius no funcionen. Això vol dir que les aplicacions que depenen d'ells per comunicar-se amb altres serveis del sistema operatiu (com el sistema de gràfics *OpenGL* o el sistema de so *Pulseaudio*) no funcionen en el nostre sistema.

El problema també apareix amb el programa d'exemple de FUSE que hem fet servir com a base, o sigui que no és degut a que haguem comés algun error en la implementació, si no que és per falta de funcionalitat de la base.

No obstant això, com que no hem trobat cap altra funcionalitat que tingués problemes a part del so i els gràfics 3D, hem decidit que no valia la pena arreglar-ho.

Resultats i conclusions dels anàlisis de programes

Visualització dels patrons de lectura d'arxius
Hem utilitzat el sistema per obtenir una llista cronològica de les lectures d'alguns arxius amb les seves posicions. Això ens permet visualitzar els patrons de lectura que tenen les aplicacions. A continuació podem veure alguns d'aquests gràfics:

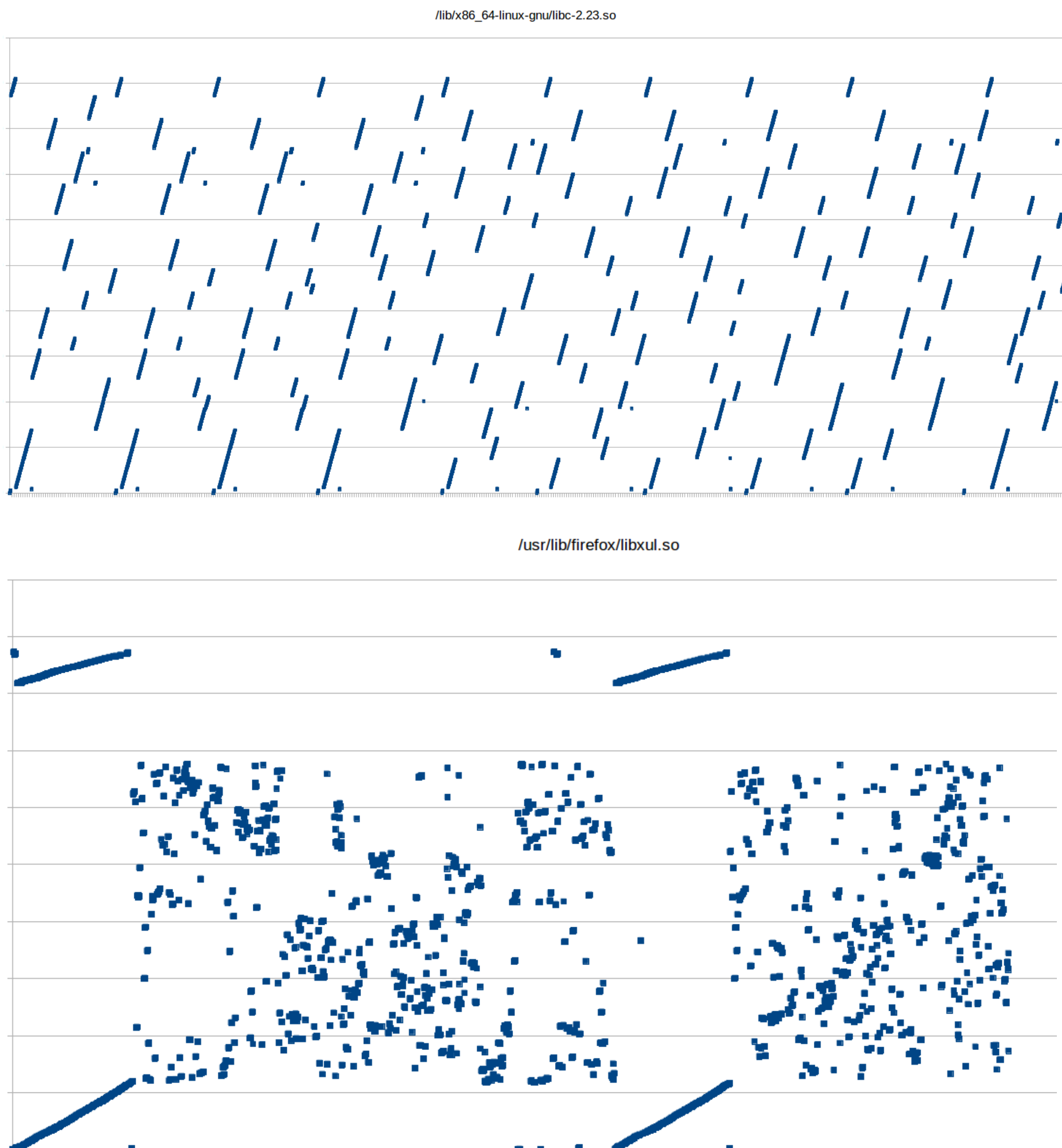

Figura 4: patrons de lectura de llibreries de Firefox

En aquests gràfics, l'eix horitzontal representa el temps i l'eix vertical representa la posició de l'arxiu. Podem veure que en el primer arxiu es produeixen lectures de blocs grans de manera més o menys aleatòria. En el segon arxiu es llegeixen primer l'inici i el final, mentre que les parts del mig es llegeixen de manera totalment aleatòria.

A continuació es mostren dos patrons de lectura molt diferents obtinguts del programa LibreOffice:

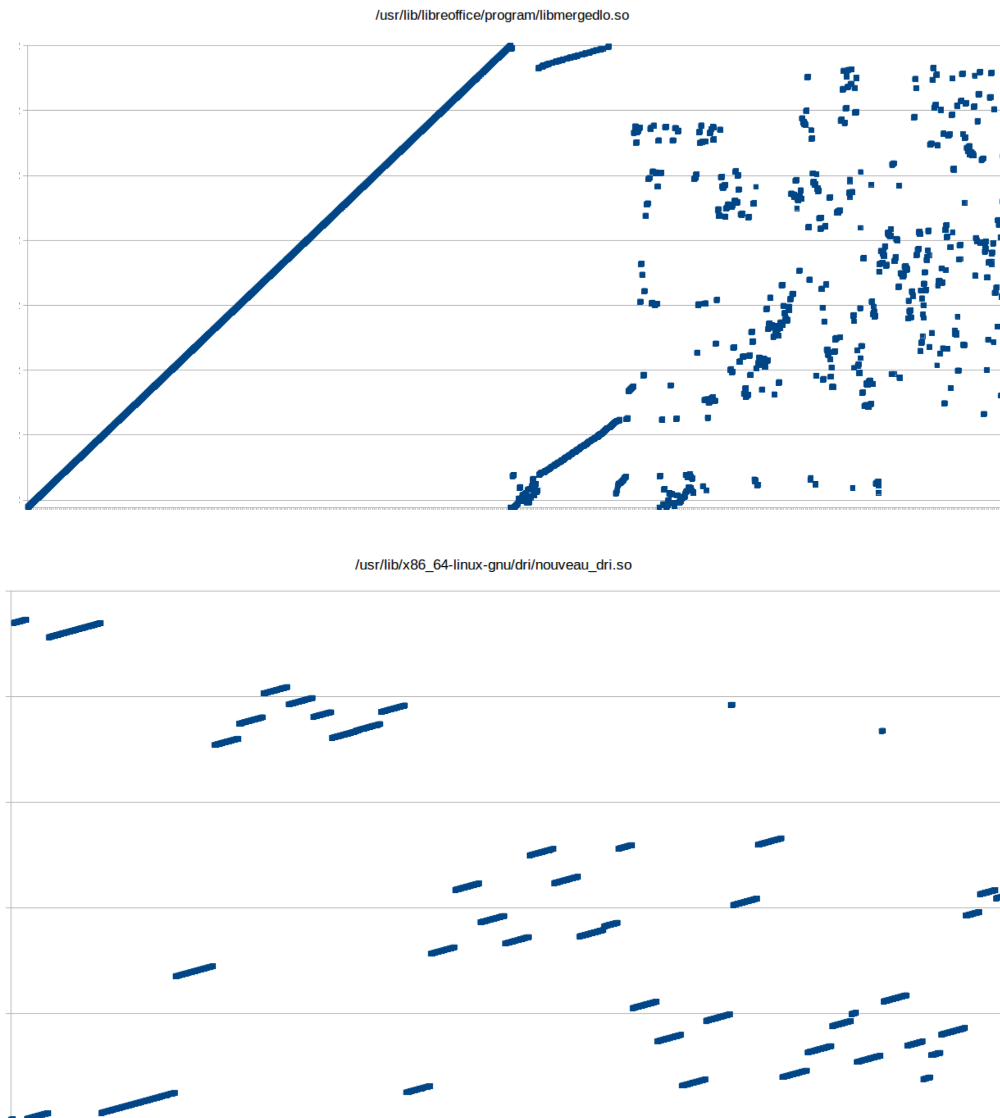

Figura 5: patrons de lectura de llibreries de LibreOffice

El primer arxiu es llegeix completament a l'inici. Aquest és el pitjor cas per al *cloudpaging,* ja que requereix enviar completament l'arxiu a l'inici. El segon arxiu, en canvi, és un cas molt més favorable ja que només es llegeixen parts dels arxius, i la majoria d'elles de manera repetida

Com a conclusió, en la majoria d'arxius sembla que el patró de lectura és bastant aleatòri.

Percentatges de blocs

Per tal d'obtenir algunes dades més objectives que ens puguin ajudar a determinar l'eficàcia del *cloudpaging*, hem creat una altra versió que imprimeix contínuament per pantalla el número de blocs diferents als quals s'ha accedit almenys un cop. Així podem observar com augmenta la quantitat de dades que s'haurien d'haver enviat per executar el programa.

Per fer la prova, hem apuntat aquestes quantitats just després d'iniciar el programa (un cop s'ha carregat completament). Després hem intentat interactuar amb el màxim de funcionalitats possible, i hem apuntat el valor final. Els resultats són:

LibreOffice:

- Inicial: 34373 blocs llegits
- Final: 84680 blocs llegits (~2,5 vegades més)

Firefox:

- Inicial: 42189 blocs llegits
- Final: 80223 blocs llegits (~1,9 vegades més)

Això vol dir que, en aquests dos casos, inicialment fan falta aproximadament la meitat de les dades que s'han fet servir al final. Tot i que aquest percentatge és tan ideal com el 20% que havíem assumit, si que sembla indicar que el cloudpaging ajudaria significativament a accelerar les descàrregues.

8.3 Gestió econòmica i temporal

Les hores finals de dedicació a cada fase del projecte han estat les següents:

Activitat	Dedicació estimada inicialment (hores)	Dedicació real
Planificació inicial	60	97
Presentació fita inicial	20	32
Anàlisi i disseny	50	19
Desenvolupament programa cloudpaging	260	87
Preparar memòria i presentació	60	220
Total	**450**	**455**

Taula 10: temps finals del projecte

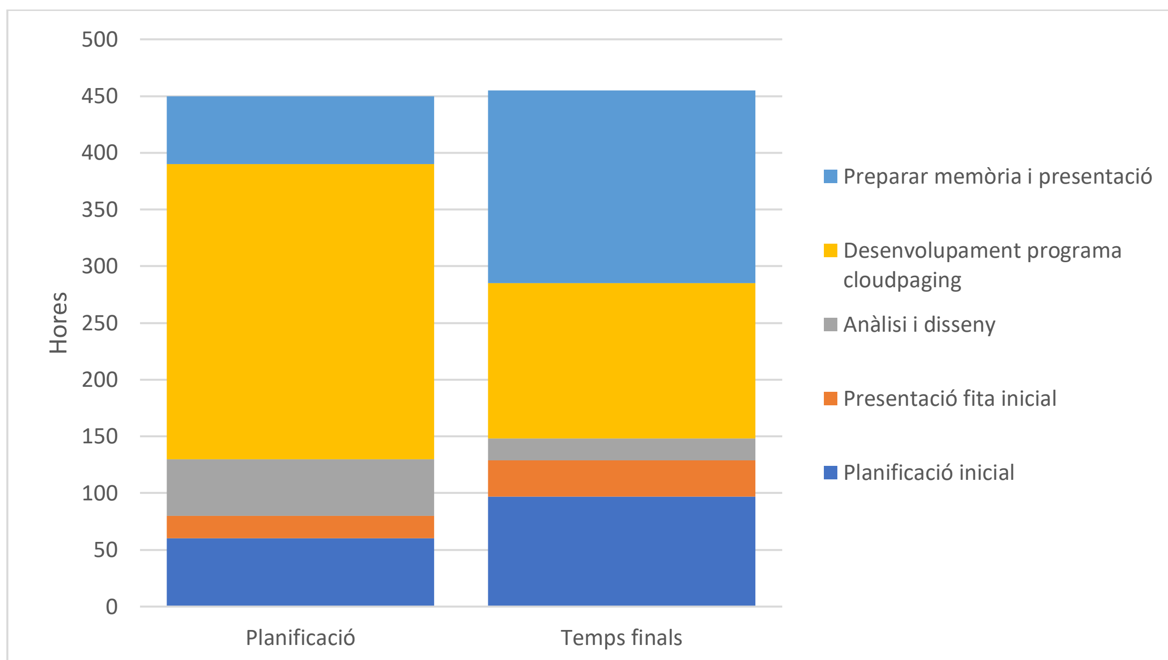

Figura 6:gràfic de les hores dedicades

Com podem veure, les hores totals han gairebé coincidit amb les inicialment previstes, però la distribució dels temps dedicats a cada fase ha estat diferent.

Aquesta desviació no s'ha degut a que haguem trobat cap problema durant la realització del treball, sinó simplement a que el programa ha estat més senzill del que s'esperava mentre que redactar ha estat més complicat. Ha estat per tant un problema de planificació.

Els costs de les fases del projecte, per tant, han estat els següents:

Fase	Cost/h	Hores reals	Cost previst	Cost real	Diferència
Planificació inicial	30	97	1.800€	2910 €	**1110 €**
Presentació fita inicial	30	32	600€	960 €	**360 €**
Anàlisis i disseny	13	19	650€	247 €	**-403 €**
Desenvolupament programa	9.4	87	2.444€	1288 €	**-1156 €**
Preparar memòria i presentació	13	220	1.800€	5100 €	**3300 €**
Total			7294 €	10504 €	**3210.8**

Taula 11: costs finals del projecte i diferències amb la planificació

Podem veure que, tot i que les hores totals s'han mantingut, el preu ha augmentat. Això és degut a que les diferents fases tenen diferents preus, i les més cares són les que han tingut un augment del pes relatiu en el projecte.

9 Conclusions

Hem assolit amb èxit els objectius del projecte, hem creat un sistema de *cloudpaging,* que permet executar programes emmagatzemats en un servidor remot al mateix temps que es descarreguen i hem comprovat que funciona.
Hem creat a més un sistema d'anàlisi de l'ús d'arxius i l'hem utilitzat per analitzar unes quantes aplicacions d'escriptori. Hem mesurat que aproximadament el 50% dels arxius no son necessaris a l'inici de l'aplicació. Tot i que aquestes proves no son estadísticament representatives de tot el software, ens permeten justificar que l'ús del cloudpaging podria accelerar la distribució d'aquestes.

No hem trobat problemes concrets durant el projecte. La metodologia ens ha funcionat com esperàvem, i la desviació de planificació ha estat per no treballar prou a l'hora de redactar.

El sistema creat està pensat com a demostració del concepte i possible base per treball futur. Per tant, ha estat desenvolupat amb l'objectiu de tenir un disseny simple i net, i no hem dedicat temps a optimitzat la velocitat del sistema.

El projecte ens ha servit per treballar el disseny i la implementació de software de sistema, planificant i dissenyant primer els programes seguint els principis d'enginyeria de software. També ens ha servit per entendre en profunditat el funcionament i la implementació dels sistemes d'arxius en Linux.

9.1 Treball futur

Degut a les limitacions de temps i recursos del projecte, hem limitat l'abast a una implementació inicial senzilla i flexible. En aquest apartat descriurem possibles millores que es podrien implementar en el futur:

Millores del sistema

Les següents funcionalitats es podrien afegir al sistema de *cloudpaging*. La majoria d'elles ja les havíem deixat explícitament fora de l'abast del projecte en el primer capítol:

- Enviament predictiu ("*prefetching*"): predir els blocs que tenen major probabilitat de ser utilitzats a continuació, i demanar-los quan la connexió estigui lliure, per millorar la velocitat del sistema.
- Els algoritmes predictius poden tenir una complexitat molt alta. Per tant, creiem que aquesta tasca podria constituir un projecte sencer.
- Una possibilitat seria crear "perfils" individuals d'aplicacions, que descriguin els patrons d'us d'aquella aplicació. Aquests perfils es podrien distribuir en els mateixos contenidors per optimitzar el procés de manera transparent a l'usuari.

- Optimitzar l’eficiència del programa: com hem explicat, en implementar el sistema ens hem centrat en crear codi senzill i flexible, i no hem dedicat temps a trobar “colls d’ampolla” que limitin la velocitat del sistema. Per tant, és probable que hi hagi moltes oportunitats per fer això.
- Afegir un algorisme de substitució de blocs. Això permetria poder executar contenidors més grans que el disc local, fent servir el mateix principi de paginació de memòria que hem descrit.
- Afegir compatibilitat amb altres sistemes operatius: el sistema només està dissenyat per Linux, però s’hauria de poder adaptar a qualsevol sistema basat en UNIX (com BSD, Android i macOS), ja que estan directament suportats per FUSE. Fins i tot podria ser possible afegir compatibilitat amb Windows, ja que també suporta sistemes d’arxius dinàmics i existeixen llibreries de compatibilitat amb FUSE [33].
- Actualment el sistema d’arxius presentat a les aplicacions és de només lectura. Idealment hauria de permetre que les aplicacions modifiquin els arxius localment. Aquestes modificacions podrien ser efímeres (existir només fins que es reiniciï el sistema), persistents localment, o ser enviades al servidor d’origen.
- Actualitzacions incrementals, és a dir, poder modificar el contenidor en el servidor i retransmetre només els blocs modificats al client. Això donaria encara més flexibilitat al les granges de servidors *cloud*.

Altres idees

- Provar d’implantar el sistema en un entorn real. Encara que el *testing* pot confirmar el funcionament d’un programa, l’única manera de trobar problemes més profunds en un sistema complex (com ara suposicions incorrectes en el disseny) és provar el seu funcionament en una feina real.
- Una idea complementaria al sistema de cloudpaging seria aprofitar el sistema d’anàlisi per trobar maneres de reduir la mida dels contenidors, trobant dades innecessàries o duplicades que es puguin eliminar.

10 Bibliografia

[1] Trefis Team, «AWS To Continue To Drive Growth For Amazon,» 25 Juliol 2017. [En línia]. Disponible: https://www.forbes.com/sites/greatspeculations/2017/07/25/aws-to-continue-to-drive-growth-for-amazon/.

[2] J. Novet, «Amazon cloud revenue soars 42 percent, topping analyst estimates,» 26 Octubre 2017. [En línia]. Disponible: https://www.cnbc.com/2017/10/26/aws-earnings-and-revenue-q3-2017.html.

[3] D. Kusnetzky, «Application streaming and why your organization should care | ZDNet,» 25 Julio 2007. [En línia]. Disponible: http://www.zdnet.com/article/application-streaming-and-why-your-organization-should-care/.

[4] N. Bunkley, «Joseph Juran, 103, Pioneer in Quality Control, Dies,» 3 Març 2008. [En línia]. Disponible: http://www.nytimes.com/2008/03/03/business/03juran.html.

[5] M. Weinberg, «Myrl Weinberg: In health-care reform, the 20-80 solution,» 27 Juliol 2009. [En línia]. Disponible: https://web.archive.org/web/20090802002952/http://www.projo.com/opinion/contributors/content/CT_weinberg27_07-27-09_HQF0P1E_v15.3f89889.html.

[6] N. Davis, «High social cost' adults can be predicted from as young as three, says study,» 12 Desembre 2016. [En línia]. Disponible: https://www.theguardian.com/science/2016/dec/12/high-social-cost-adults-can-be-identified-from-as-young-as-three-says-study.

[7] P. Rooney, «Microsoft's CEO: 80-20 Rule Applies To Bugs, Not Just Features,» 3 October 2002. [En línia]. Disponible: http://www.crn.com/news/security/18821726/microsofts-ceo-80-20-rule-applies-to-bugs-not-just-features.htm.

[8] J. Bird, «Applying the 80:20 Rule in Software Development,» 14 Novembre 2013. [En línia]. Disponible: http://swreflections.blogspot.com.es/2013/11/applying-8020-rule-in-software.html.

[9] B. Wescott, «Chapter 3: Useful laws,» de *The Every Computer Performance Book*, CreateSpace, 2013.

[10] S. K. Chang, de *Data Structures and Algorithms*, World scientific, 2003, p. 19.

[11] Citrix Systems, Inc, «Application Streaming Overview,» 29 April 2016. [En línia]. Disponible: https://docs.citrix.com/pt-br/app-streaming/6-7/ps-stream-intro-wrapper-for-xenapp-library-v2.html.

[12] B. Botelho, «Citrix kills its Application Streaming feature, directs users to App-V,» 22 May 2013. [En línia]. Disponible: http://searchvirtualdesktop.techtarget.com/news/2240184559/Citrix-kills-its-Application-Streaming-feature-directs-users-to-App-V.

[13] Microsoft, «Microsoft Application Virtualization System Features,» 4 05 2017. [En línia]. Disponible: https://docs.microsoft.com/en-us/microsoft-desktop-optimization-pack/appv-v4/overview-of-application-virtualization#microsoft-application-virtualization-system-features.

[14] Micro Focus, «Features - Desktop Containers,» 2017. [En línia]. Disponible: https://www.microfocus.com/es-es/products/desktop-containers/features/.

[15] Numecent, «Cloudpaging - Numecent,» 2017. [En línia]. Disponible: https://www.numecent.com/cloudpaging/.

[16] Microsoft, «Overview of Click-to-Run,» 25 08 2016. [En línia]. Disponible: https://technet.microsoft.com/en-us/library/jj219427.aspx.

[17] Y. O. ,. H. K. Sunwook Kim, «On-demand Software Streaming System for Embedded System,» Department of Computer Science and Engineering, Daejeon, Korea, 2006.

[18] H. e. a. «Software caching using dynamic binary rewriting for embedded devices,» de *Int'l conference on Parallel Processing*, 2002.

[19] P. Kuachroen, «Embedded software streaming via block streaming,» Georgia Institute of Technology, 2004.

[20] M. e. a. de *Java Virtual Machine*, Cambridge, MA, O'Reilly, 1997, pp. 44-45.

[21] H. e. a. «Liquid software: a new paradigm for networked systems,» Tech Rep. 96-11, Dept. of Computer Science, Arizona, 1996.

[22] Virtuozzo, «Virtuozzo 7 Platform Data Sheet,» Noviembre 2016. [En línia]. Disponible: https://virtuozzo.com/wp-content/uploads/vz7/doc/Virtuozzo7_Platform_DS_EN_A4_20161103.pdf .

[23] A. Reber, «Container Live Migration Using runC and CRIU,» 8 Diciembre 2016. [En línia]. Disponible: http://rhelblog.redhat.com/2016/12/08/container-live-migration-using-runc-and-criu/.

[24] PageGroup, «Estudio de remuneración 2017 - tecnología,» 2017. [En línia]. Disponible: https://www.pagepersonnel.es/sites/pagepersonnel.es/files/PG_ER_IT.pdf.

[25] Generalitat de Catalunya, «Factors d'emissió associats a l'energia,» [En línia]. Disponible:

http://canviclimatic.gencat.cat/ca/redueix_emissions/factors_demissio_associats_a_lenergia/index.html.

[26] European environment agency, «Greenhouse gas emissions per capita,» 2015. [En línia]. Disponible: http://ec.europa.eu/eurostat/tgm/table.do?tab=table&init=1&language=en&pcode=t2020_rd300&plugin=1.

[27] D. Bauer y K. Papp, «Book Review Perspectives: The Jevons Paradox and the Myth of Resource Efficiency Improvements,» *Sustainability: Science, Practice, & Policy,* vol. 5, nº 1, 18 Març 2009.

[28] R. York, «Ecological paradoxes: William Stanley Jevons and the paperless office,» *Human Ecology Review,* vol. 13, nº 2, p. 143–147, 2006.

[29] libfuse, «Libfuse wiki,» 18 11 2017. [En línia]. Disponible: https://github.com/libfuse/libfuse/wiki/Filesystems.

[30] Tuxera, «NTFS-3G,» [En línia]. Disponible: https://www.tuxera.com/community/open-source-ntfs-3g/.

[31] A. Hitomi, Y. Ahiska y B. Ahiska, «Adaptive cloud-based application streaming». US Patente US 20150127774 A1, 7 May 2015.

[32] IEEE and The Open Group, «The Open Group Base Specifications Issue 7, 2018 edition,» 2018. [En línia]. Disponible: http://pubs.opengroup.org/onlinepubs/9699919799/functions/hcreate.html.

[33] Dokan, «Dokan,» 2018. [En línia]. Disponible: https://dokan-dev.github.io/.

11 Annex 1: Diagrama de Gantt