UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Optimización de arquitecturas distribuidas para el procesado de datos masivos

José Herrera Hernández

Dirigida por: Dr. Germán Moltó Martínez

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Tesis por compendio de publicaciones
para la obtención del
Grado de Doctor

Julio 2020

"Si piensas que vales lo que sabes, estás muy equivocado.
Tus conocimientos de hoy no tienen mucho valor más allá de un par de años.
Lo que vales es lo que puedes llegar a aprender,
la facilidad con la que te adaptas a los cambios que esta profesión nos regala tan frecuentemente."

José M. Aguilar, en *'Cómo tu blog te ayuda a encontrar empleo'*

"Nunca se es demasiado mayor para aprender más de lo que se sabe
y llegar a ser capaz de hacer más de lo que ya se puede."

Isaac Asimov en *'Los límites de la fundación'*

"En el estudio no existe la saciedad."

Erasmo de Róterdam

Agradecimientos

Quisiera agradecer a todas las personas que han estado a mi lado durante estos años de duro trabajo y en especial a mi familia, que ha sido un gran apoyo durante los momentos de decaimiento que es cuando más se ha hecho necesario su aliento y comprensión.

Debo agradecer a todo el claustro del Máster Universitario en Computación Paralela y Distribuida de la Universitat Politècnica de València que han sabido inducir la necesidad de la investigación que llevaba dentro y que permanecía oculta detrás de capas producidas por años de burocrático trabajo.

A Ignacio Blanquer, que me acogió como mi primer director de los estudios de doctorado y que confió en mis posibilidades, guiándome hacia mis preferencias y resaltando mis capacidades.

Adshuhiro Takasu como *Sensei* en mi estancia en el *National Institut of Informatics* de Tokyo que, durante un breve pero intenso período de tiempo, me guió en los métodos y modelos de investigación multidisciplinar en ambientes internacionales y que me permitió una estancia enriquecedora fuera de las vías universitarias existentes más frecuentes.

Y por supuesto, por último y por ello no menos importante, a mi director de tesis, Germán Moltó, que confió en mí, empleando horas de trabajo para encaminarme hacia una carrera investigadora que resulta enriquecedora y hacerme comprender los valores de este complejo mundo de la búsqueda de nuevos horizontes y que espero que, en un futuro cercano, me de muchas satisfacciones.

Gracias a todos.

Resumen

La utilización de sistemas para el tratamiento eficiente de grandes volúmenes de información ha crecido en popularidad durante los últimos años. Esto conlleva el desarrollo de nuevas tecnologías, métodos y algoritmos, que permitan un uso eficiente de las infraestructuras. El tratamiento de grandes volúmenes de información no está exento de numerosos problemas y retos, algunos de los cuales se tratarán de mejorar. Dentro de las posibilidades actuales debemos tener en cuenta la evolución que han tenido los sistemas durante los últimos años y las oportunidades de mejora que existan en cada uno de ellos.

El primer sistema de estudio, el Grid, constituye una aproximación inicial de procesamiento masivo y representa uno de los primeros sistemas distribuidos de tratamiento de grandes conjuntos de datos. Participando en la modernización de uno de los mecanismos de acceso a los datos se facilita la mejora de los tratamientos que se realizan en la genómica actual. Los estudios que se presentan están centrados en la transformada de Burrows-Wheeler, que ya es conocida en el análisis genómico por su capacidad para mejorar los tiempos en el alineamiento de cadenas cortas de polinucleótidos. Esta mejora en los tiempos, se perfecciona mediante la reducción de los accesos remotos con la utilización de un sistema de caché intermedia que optimiza su ejecución en un sistema Grid ya consolidado. Esta caché se implementa como complemento a la librería de acceso estándar GFAL utilizada en la infraestructura de IberGrid.

En un segundo paso se plantea el tratamiento de los datos en arquitecturas de Big Data. Las mejoras se realizan tanto en la arquitectura Lambda como Kappa mediante la búsqueda de métodos para tratar grandes volúmenes de información multimedia. Mientras que en la arquitectura Lambda se utiliza Apache Hadoop como tecnología para este tratamiento, en la arquitectura Kappa se utiliza Apache Storm como sistema de computación distribuido en tiempo real. En ambas arquitecturas se amplía el ámbito de utilización y se optimiza la ejecución mediante la aplicación de algoritmos que mejoran los problemas en cada una de las tecnologías.

El problema del volumen de datos es el centro de un último escalón, por el que se permite mejorar la arquitectura de microservicios. Teniendo en cuenta el número total

de nodos que se ejecutan en sistemas de procesamiento tenemos una aproximación de las magnitudes que podemos obtener para el tratamiento de grandes volúmenes. De esta forma, la capacidad de los sistemas para aumentar o disminuir su tamaño permite un gobierno óptimo. Proponiendo un sistema bioinspirado se aporta un método de autoescalado dinámico y distribuido que mejora el comportamiento de los métodos comúnmente utilizados frente a las circunstancias cambiantes no predecibles.

Las tres magnitudes clave del Big Data, también conocidas como V's, están representadas y mejoradas: velocidad, enriqueciendo los sistemas de acceso de datos por medio de una reducción de los tiempos de tratamiento de las búsquedas en los sistemas Grid bioinformáticos; variedad, utilizando sistemas multimedia menos frecuentes que los basados en datos tabulares; y por último, volumen, incrementando las capacidades de autoescalado mediante el aprovechamiento de contenedores software y algoritmos bioinspirados.

Resum

La utilització de sistemes per al tractament eficient de grans volums d'informació ha crescut en popularitat durant els últims anys. Açò comporta el desenvolupament de noves tecnologies, mètodes i algoritmes, que aconsellen l'ús eficient de les infraestructures. El tractament de grans volums d'informació no està exempt de nombrosos problemes i reptes, alguns dels quals es tractaran de millorar. Dins de les possibilitats actuals hem de tindre en compte l'evolució que han tingut els sistemes durant els últims anys i les ocasions de millora que existisquen en cada un d'ells.

El primer sistema d'estudi, el Grid, constituïx una aproximació inicial de processament massiu i representa un dels primers sistemes de tractament distribuït de grans conjunts de dades. Participant en la modernització d'un dels mecanismes d'accés a les dades es facilita la millora dels tractaments que es realitzen en la genòmica actual. Els estudis que es presenten estan centrats en la transformada de Burrows-Wheeler, que ja és coneguda en l'anàlisi genòmica per la seua capacitat per a millorar els temps en l'alineament de cadenes curtes de polinucleòtids. Esta millora en els temps, es perfecciona per mitjà de la reducció dels accessos remots amb la utilització d'un sistema de memòria cau intermèdia que optimitza la seua execució en un sistema Grid ja consolidat. Esta caché s'implementa com a complement a la llibreria d'accés estàndard GFAL utilitzada en la infraestructura d'IberGrid.

En un segon pas es planteja el tractament de les dades en arquitectures de Big Data. Les millores es realitzen tant en l'arquitectura Lambda com a Kappa per mitjà de la busca de mètodes per a tractar grans volums d'informació multimèdia. Mentre que en l'arquitectura Lambda s'utilitza Apache Hadoop com a tecnologia per a este tractament, en l'arquitectura Kappa s'utilitza Apache Storm com a sistema de computació distribuït en temps real. En ambdós arquitectures s'àmplia l'àmbit d'utilització i s'optimitza l'execució per mitjà de l'aplicació d'algoritmes que milloren els problemes en cada una de les tecnologies.

El problema del volum de dades és el centre d'un últim escaló, pel qual es permet millorar l'arquitectura de microserveis. Tenint en compte el nombre total de nodes que s'executen en sistemes de processament tenim una aproximació de les magnituds

que podem obtindre per al tractaments de grans volums. D'aquesta manera la capacitat dels sistemes per a augmentar o disminuir la seua dimensió permet un govern òptim. Proposant un sistema bioinspirat s'aporta un mètode d'autoescalat dinàmic i distribuït que millora el comportament dels mètodes comunment utilitzats enfront de les circumstàncies canviants no predictibles.

Les tres magnituds clau del Big Data, també conegudes com V's, es troben representades i millorades: velocitat, enriquint els sistemes d'accés de dades per mitjà d'una reducció dels temps de tractament de les busques en els sistemes Grid bioinformàtics; varietat, utilitzant sistemes multimèdia menys freqüents que els basats en dades tabulars; i finalment, volum, incrementant les capacitats d'autoescalat per mitjà de l'aprofitament de contenidors i algoritmes bioinspirats.

Abstract

The use of systems for the efficient treatment of large data volumes has grown in popularity during the last few years. This has led to the development of new technologies, methods and algorithms to efficiently use of infrastructures. The Big Data treatment is not exempt from numerous problems and challenges, some of which will be attempted to improve. Within the existing possibilities, we must take into account the evolution that systems have had during the last years and the improvement that exists in each one.

The first system of study, the Grid, constitutes an initial approach of massive distributed processing and represents one of the first treatment systems of big data sets. By researching in the modernization of the data access mechanisms, the advance of the treatments carried out in current genomics is facilitated. The studies presented are centred on the Burrows-Wheeler Transform, which is already known in genomic analysis for its ability to improve alignment times of short polynucleotids chains. This time, the update is enhanced by reducing remote accesses by using an intermediate cache system that optimizes its execution in an already consolidated Grid system. This cache is implemented as a GFAL standard file access library complement used in IberGrid infrastructure.

In a second step, data processing in Big Data architectures is considered. Improvements are made in both the Lambda and Kappa architectures searching for methods to process large volumes of multimedia information. For the Lambda architecture, Apache Hadoop is used as the main processing technology, while for the Kappa architecture, Apache Storm is used as a real time distributed computing system. In both architectures the use scope is extended and the execution is optimized applying algorithms that improve problems for each technology.

The last step is focused on the data volume problem, which allows the improvement of the microservices architecture. The total number of nodes running in a processing system provides a measure for the capacity of processing large data volumes. This way, the ability to increase and decrease capacity allows optimal governance. By

proposing a bio-inspired system, a dynamic and distributed self-scaling method is provided improving common methods when facing unpredictable workloads.

The three key magnitudes of Big Data, also known as V's, will be represented and improved: speed, enriching data access systems by reducing search processing times in bioinformatic Grid systems; variety, using multimedia data less used than tabular data; and finally, volume, increasing self-scaling capabilities using software containers and bio-inspired algorithms.

Tabla de Contenidos

Lista de Figuras

Lista de Tablas

Nomenclatura

Símbolos Griegos

$\Delta - \delta$ Delta - Cuarta letra del alfabeto griego.

$\Lambda - \lambda$ Lambda - Undécima letra del alfabeto griego.

$K - \kappa$ Kappa - Décima letra del alfabeto griego.

$Z - \zeta$ Zeta - Sexta letra del alfabeto griego.

Acrónimos / Abreviaciones

ACK Acknowledge.

API Application Programming Interface.

$BDII$ Berkley Database Information Index.

BWT Burrows-Wheeler Transform.

$CaaS$ Container as a Service.

CEO Chief Executive Officer - Director Ejecutivo.

DAG Grafo Acíclico Dirigido.

DTW Dynamic Time Warping.

EAI Alianza Europea para la Innovación.

EGI European Grid Infraestructure.

EID Escuela Internacional de Doctorado.

$ES - NGI$ National Grid Initiative de España.

ESB Enterprise Service Bus.

EVS Explained Variance Score.

$GFAL$ Grid File Access Library.

GFS Google File System.

$HDFS$ Hadoop Distributed File System.

HPC High Performance Computing.

IA Inteligencia Artificial.

$INCD$ Infraestrutura Nacional de Computação Distribuida.

IoT Internet of Things.

JCR Journal Citation Report.

$LRMS$ Local Resource Management System.

MAE Mean Absolute Error.

NGI National Grid Initiative.

OGF Open Grid Forum.

$OGSA$ Open Grid Service Architecture.

$OpenIMAJ$ Open Intelligent Multimedia Analysis.

PCM Pulse Coded Modulation.

PoC Proof of Concept.

$R2S$ Coefficient of Determination o R-Squared value.

$RANSAC$ RANdom SAmple Consensus.

$REST$ Representational state transfer.

$RMSE$ Root Mean Squared Error.

$S4$ Simple Scalable Streaming System.

SE Storage Element.

$SIFT$ Scale-Invariant Feature Transform.

$SIMD$ Simple Instruction Multiple Data.

SOA Service Oriented Architecture.

$SOAP$ Service Object Access Protocol.

$URJC$ Universidad Rey Juan Carlos.

$VSDL$ Web Service Description Language.

WN Working Node.

$WSRF$ Web Service Resource Framework.

$YARN$ Yet Another Resource Negotiator.

Capítulo 1

Introducción

En la actualidad nadie es ajeno, o al menos no expone ninguna duda, sobre el valor que tiene el procesamiento de grandes volúmenes de datos. La que se está llamando *cuarta revolución industrial* [214] tiene como objetivo, no la creación de bienes materiales como era la meta de las revoluciones anteriores, sino la creación de conocimiento mediante la transformación digital y la obtención de valor de una forma automatizada.

Por esta razón, no es extraño que en los medios de comunicación se destaquen aquellos titulares en los que están involucrados empresas generadoras de datos, firmas tecnológicas o compañías de servicios analíticos, todas ellas denominadas *data-driven organizations*. Este interés es debido a la posición destacada del dato dentro de la cadena de valor [25]. El dato ha pasado de ser un residuo indeseado de un proceso de transformación al mayor objeto de deseo entre las empresas que monetizan la información.

Relacionadas con esto, hemos encontrado grandes corporaciones que han sabido administrar aquellos datos infrautilizados para lograr una nueva economía basada en una mejora competitiva del modelo de negocio. Como indica Gaskell [92] *"Los datos deben estar en el centro de todo lo que se hace"*. El secreto subyacente se basa en no descartar las pequeñas cantidades de información que proporcionan los usuarios o dispositivos sino lograr que estas pequeñas porciones de información sumen hacia un objetivo final. Esta acumulación de pequeñas porciones lleva al aumento exponencial del volumen de datos a transportar, procesar, interpretar y almacenar. Se deben considerar los datos brutos como precursores de la información necesaria para la toma de decisiones.

Unos de los aspectos esenciales de estos procesos de obtención, tratamiento y almacenamiento es la velocidad [88]. En función de la mayor velocidad del proceso, mejor respuesta ante los cambios en el mercado y aumento de la ventaja competitiva

en el modelo de negocio aplicado. Se ha pasado de ser usuarios ocasionales de datos, sin importar el momento en el que se generan, a consumidores ávidos de la información generada en tiempo real y por múltiples canales.

A consecuencia de esta necesidad, el reto al que nos enfrentamos es la mecanización del tratamiento de grandes volúmenes de información ya que no se puede concretar una técnica común para todos los casos. Los esfuerzos que se han ido realizando a lo largo de los últimos años han pasado por el empleo de numerosas arquitecturas, técnicas y métodos. Algunos de estos han llegado a suponer una mejora de los aspectos esenciales del procesamiento pero siempre han surgido nuevas necesidades a las que ha sido imprescindible responder con nuevos planteamientos.

Ian Foster, Carl Kesselman y Steve Tuecke, conocidos como los padres del Grid, realizaron los esfuerzos, al final de los noventa y principios del 2000, para lograr infraestructuras que permitieran el acceso rápido a la gran potencia computacional de los grandes sistemas de la época, además de aunar los métodos operativos comunes que debían regir la colaboración entre instituciones. Ian Foster y Carl Keselman publicaron en 1999 el trabajo seminal en esta materia *"The Grid: Blueprint for a new computing infrastructure"* [66] que podemos considerar como la línea de salida en la carrera por el procesamiento avanzado de grandes volúmenes de datos mediante sistemas geográficamente distribuidos.

Para ayudarnos en este procesamiento masivo se han caracterizado tres paradigmas que reflejan los modos habituales de ejecución de las tareas: *batch*, *stream processing* y tiempo real. Aunque los sistemas de tiempo real suponen un tipo de sistemas donde se asegura la duración de procesamiento máximo y se encuentran muy cercanos a los sistemas interactivos [218], en el resto de este documento se consideran los sistemas de *streaming* y de tiempo real como equivalentes, debido a que el principal propósito que debe guiar a aquellos sistemas de flujo continuo de información debe ser la respuesta de una forma constante y con un retardo óptimo.

El primer tipo, *"batch processing"*, es el procesamiento que trata bloques de datos que han sido almacenados en un período de tiempo anterior. Habitualmente estos sistemas, dentro del ámbito del procesamiento masivo de datos, están asentados en el paradigma MapReduce [45] aunque es común utilizar otros modelos. La aproximación más utilizada para este tipo de tecnología es utilizar Apache Hadoop [70] y HDFS (Hadoop Distributed File System) [76]. En la actualidad son tecnologías muy probadas, robustas y que permiten alcanzar el nivel de procesamiento al que aspiran los arquitectos del Big Data para sus proyectos.

El paradigma de *"streaming processing"*, a diferencia del enfoque anterior, funciona en tiempo real o semi-real. Las fuentes de datos, que proporcionan flujos constantes, suministran la materia prima para un proceso que debe funcionar de forma continua e indefinidamente. Aun así, dentro de nuestra categoría, podemos encontrar matices de cómo se logran estos fines pudiendo acuñar términos como *full streaming*, donde encontramos soluciones tecnológicas como Apache Storm [83], Apache Samza [80] o Apache Flink [74], o bien *micro-batch* con herramientas como Spark Streaming [81] o Storm Trident [84].

En este punto Nathan Marz publicó en su libro [171]: *"La arquitectura Lambda resuelve el problema de calcular funciones arbitrarias en tiempo real descomponiendo el problema en tres capas: la capa batch, la capa de servicio y la capa de velocidad"*.

Aunque los sistemas de contenedores aparecen previamente, en 2013 emergió Docker [122]. El crecimiento en su uso lo ha hecho uno de los más populares gestores de contenedores existentes, convirtiéndose rápidamente en uno de los estándares de la industria. En la Figura 1.1 se puede observar el momento de su aparición junto con un diagrama temporal de los años de aparición de algunas tecnologías y arquitecturas relativas a los temas tratados.

Ya en 2014 Jay Kreps acuñó el término arquitectura Kappa [144] orientada a la analítica solo en *streaming*, eludiendo el almacenamiento, minimizando tiempos, evitando el recómputo en la capa *batch* y permitiendo a la capa de *streaming* procesar los cambios si existe alguna modificación de la lógica de negocio.

En 2015 los analistas de Gartner eliminaron del *Hype Cycle for Emerging Techonologies report* el término "Big Data" [236] debido principalmente a su madurez en los desarrollos y su nivel de explotación en los tratamientos.

No comprenderíamos en su totalidad la magnitud del estudio sin conocer la evolución tecnológica temporal de los sistemas objeto de observación. Los expertos apuntan a un volumen total de datos cercano a los 175 Zettabytes en 2025 que supone un aumento del 530% respecto a 2018 [201][36]. Uno de los factores que consolida este crecimiento es el rápido aumento en la generación de datos procedentes de individuos y empresas por igual, afianzado por la incorporación de la inteligencia artificial como medio de análisis. Por ello es necesario plantear una visión dinámica de estas arquitecturas, no solo en su capacidad para adaptarse al tipo de trabajo realizado sino también al volumen esperado o demandado. La administración dinámica, que depende en gran medida de la heterogeneidad de los datos, se puede alcanzar facilitando al mismo tiempo la robustez y fiabilidad al modelo seleccionado. El autoescalado es la respuesta

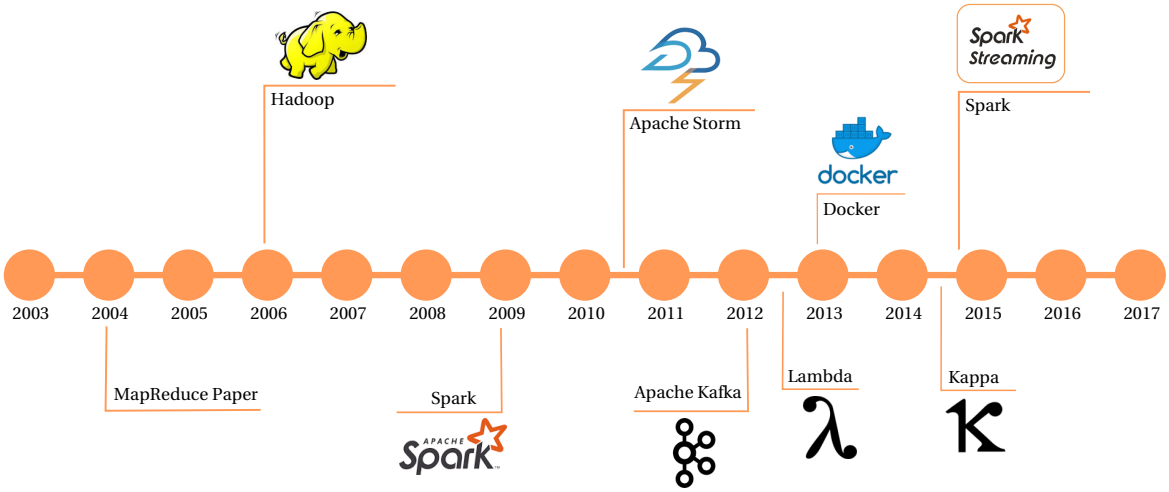


Fig. 1.1 Evolución histórica simplificada de la tecnología de Big Data.

gestionada por sí mismo de los sistemas Big Data a la necesidad de procesamiento de volumen variable.

Esto conduce a la inevitable y continua evolución de los sistemas de procesamiento a la cual nos debemos adaptar sin demora. De forma similar a como *DevOps* [46] ha llevado a los procesos de desarrollo de software a la reducción de los períodos de entrega, podemos ver esta analogía en el conjunto de sistemas actuales, donde la evolución llega con una gran velocidad, y las mejoras debidas a la aplicación de nuevos conceptos se deben producir en períodos de tiempo reducidos. Estamos en un momento donde los períodos de incorporación de nuevas mejoras en arquitecturas, sistemas y modelos de tratamiento también se reducen debido al desarrollo competitivo, y por tanto económico, que puede obtenerse.

1.1 Motivación

En el marco de la realización de los documentos publicados, debemos conocer que durante los últimos años se ha producido una considerable evolución dentro de los sistemas para el procesamiento de grandes volúmenes de información. El extraordinario interés que existe sobre estos modelos de tratamientos de datos hace que los trabajos de investigación en este área también sean atractivos. La motivación económica es un gran acicate para la evolución tecnológica. Una motivación basada en la reducción de costos, pero también en el modelo de la *"Data Driven Organization"* [152], hace que no solo se presenten estos sistemas como un ejercicio teórico pendiente de resolver sino que cale profundamente en los ámbitos de la economía y la sociedad, entre otros. Todo

trabajo de investigación está alentado por una necesidad de mejorar determinadas áreas de trabajo. El conjunto de esta tesis esta justificado por algunas motivaciones que resultan diferentes pero complementarias, a la vez que son altamente dependientes del desarrollo técnico en otras áreas de investigación no propias del Big Data.

Debemos tener en cuenta la necesidad de disponer de modelos adecuados para el tratamiento de datos para grandes volúmenes. Los algoritmos paralelos o distribuidos son una gran fuente de información para este tipo de tratamiento pero no de forma exclusiva. Tenemos estudios de aplicabilidad especialmente dedicados a algoritmos de Big Data en bioinformática [97] [211], clustering [53] [12], inteligencia artificial (IA) [47], indexación y búsquedas, pronóstico [102], aplicabilidad al streaming y análisis de cuestiones éticas entre muchas más.

Podemos tratar el uso de infraestructuras capaces de ejecutar trabajos que utilicen grandes volúmenes de información desde varios puntos de vista: haciéndolas más sostenibles [237], reduciendo los costes de operación [57] [10], mejorando su mantenimiento y/o aumentando su disponibilidad en un período de tiempo.

Durante los últimos años, se ha tomado consciencia de la necesidad de mejorar la respuesta al aumento de la volumetría de los procesos a desarrollar, incentivado especialmente por el incremento de la información existente y los resultados que se desean obtener pero, al mismo tiempo, han aparecido otros condicionantes no tan obvios como pueden ser el ahorro energético, la computación verde, la huella de carbono o los sistemas sostenibles.

Por ese mismo impulso común, hay que tener en cuenta el avance necesario en la incorporación de nuevas variedades de información para mejorar los sistemas, estimulado por la aparición de nuevos modelos de aplicabilidad como pueden ser el IoT (Internet of Things), la inteligencia artificial o la necesidad de tratar información multimedia.

Existe también una demanda constante por el aumento de la velocidad en el tratamiento de los datos, requiriéndose sistemas más eficaces y eficientes que sean capaces de manipular los datos a gran velocidad o incluso ofrecer de forma continua las informaciones tratadas.

Son justamente estas tres últimas motivaciones las que hacen que los atributos de velocidad, variedad y volumen, las tres V's propias de un sistema Big Data [150], mejoren en su conjunto, logrando sistemas más capaces y a la vez eficientes.

1.2 Objetivos de la tesis

Aunque las tecnologías han variado durante los años de realización de esta Tesis Doctoral, siempre han existido unas líneas de investigación comunes a todas ellas. Los objetivos generales no se han centrado en una única herramienta, sino que han evolucionado dependiendo del estado de desarrollo técnico disponible en cada momento, permitiendo la flexibilidad necesaria para ser enfocado en las necesidades existentes.

Posteriormente, en base a las líneas guía generales, sí que se han fijado objetivos específicos en cada uno de los desarrollos particulares, centrados ya en tecnologías concretas. En estos casos particulares ya se han generado hipótesis de trabajo que eran confirmadas o rebatidas por los modelos desarrollados.

Los objetivos se plantean basándose en unos antecedentes que fortalecen la investigación realizada:

- Creciente evolución de los sistemas de procesamiento de datos, tanto en número como en volumen tratado.
- Incremento contrastado de las aportaciones técnicas dentro del ámbito estudiado.
- Aumento de las necesidades actuales de procesamiento de datos.
- Interés de la sociedad por el tratamiento y utilización de la información.

La principal hipótesis que podemos realizar alrededor del tema de la tesis es que los sistemas actuales relacionados con el mundo del Big Data, poseen numerosas áreas de mejora que pueden ser explotadas. Por un lado tenemos la posibilidad de mejorar los algoritmos, estudiándolos y realizando aportaciones que optimicen su comportamiento. Por otro lado podemos observar cómo las infraestructuras se adaptan a los tratamientos de datos poco utilizados y también podemos tener en cuenta cómo estas estructuras se pueden mejorar con el fin de trabajar con un volumen de información creciente.

El **objetivo principal**, que se formuló como eje conductor de todo el estudio, es *"realizar mejoras en las arquitecturas distribuidas que se utilizan en el procesamiento de datos masivos contribuyendo a su desarrollo técnico"*. De igual forma, como **objetivo secundario**, se trata de *"desarrollar o probar nuevos componentes que enriquezcan la arquitectura de los entornos tecnológicos que se están utilizando de forma que se mejoren los atributos propios de los sistemas de tratamiento"*. En definitiva, aportar características técnicas como la elasticidad del sistema, facilitar el acceso de los procesos a los datos remotos o la utilización de tipos variados de datos en arquitecturas altamente distribuidas.

Se han creado unos objetivos transversales que subyacen, en la medida de lo posible, en las cuatro contribuciones científicas realizadas y que son los siguientes:

- **Objetivo Transversal 1**. Mejorar los sistemas de procesamiento de grandes volúmenes de información, identificando las áreas de trabajo, priorizando aquellos segmentos con mayores posibilidades, necesidades y/o oportunidades.
- **Objetivo Transversal 2**. Analizar el estado del arte de las infraestructuras para establecer la aportación más adecuada, adaptando la investigación al estado del arte.
- **Objetivo Transversal 3**. Contribuir al corpus técnico de la tecnología seleccionada. Independientemente de las necesidades de publicación que toda investigación debería tener, se ha incluido, como parte crucial, la necesidad de transmitir los conocimientos adquiridos a la comunidad científica y más específicamente con cada una de las herramientas utilizadas.
- **Objetivo Transversal 4**. Presentar casos de uso o modelos de estudio como herramientas para explorar las exigencias de los sistemas actuales y como medio para validar las hipótesis realizadas.
- **Objetivo Transversal 5**. Probar y evaluar al menos una mejora en cada uno de los ámbitos seleccionados. Es necesario que todas las tecnologías seleccionadas obtengan una aportación para no quedarse en el mero estudio del estado del arte mencionado en objetivos transversales anteriores.

1.3 Justificación de los estudios

Desde el inicio de las actividades que se analizan en este documento, se ha tenido en cuenta que la definición de algunos objetivos del doctorado no podrían mantenerse inmutables durante todo el período y que deberían adaptarse en función de la evolución tecnológica, en cada uno de los momentos de inicio de las tareas parciales. La capacidad de incorporar las evoluciones de los entornos debía ser uno de los objetivos transversales que se mantuviera durante todo el desarrollo.

Del mismo modo que con el transcurso del tiempo se ha centrado el estudio de la investigación a un ámbito de trabajo más reducido, también se ha realizado una tarea de conceptualización. Se ha evolucionado de un problema más individual que podía mejorar una incógnita en un ámbito más concreto a uno más abstracto, y que era aplicable de forma conceptual a más sistemas.

Al inicio, se planteó como fundamental la tarea de reconocer los entornos de trabajo que se querían mejorar y la vía para hacerlo era explorar las opciones que se planteaban, profundizando en las áreas de investigación dentro del propio entorno. De esta forma, se evaluaron opciones que permitían mantener un marco de trabajo amplio que iría reduciéndose durante los años siguientes en base a los conocimientos que se tenían y la evolución del estado del arte.

Como se ha comentado anteriormente, las herramientas para el tratamiento de grandes volúmenes de datos han evolucionado rápidamente desde los modelos Grid hasta las arquitecturas de microservicios, pasando por arquitecturas Lambda o Kappa. Todo ello facilitado por el elevado nivel de implantación tecnológica y debido a los beneficios que los nuevos sistemas ofrecían. El impacto que tienen estos cambios tecnológicos no es solamente visible en los contenidos de este estudio sino que también es apreciable en las decisiones y estrategias que deben asumir las organizaciones con base tecnológica.

La tesis la podemos dividir en tres retos de investigación correspondientes a las tres dimensiones o V's del Big Data (Velocidad, Variedad y Volumen). Cada uno de los retos posee sus propios objetivos y una o varias publicaciones que lo secundan, todo ello utilizando los mismos objetivos transversales y generales ya expuestos.

El reto inicial corresponde a la primera "V" o dimensión de Velocidad. En él encontramos, como herramienta para el tratamiento de grandes volúmenes de datos, al Grid que fue el centro de los posibles avances. Esto llevó a la realización de un estudio sobre la mejora del procedimiento de alineación de cromosomas, que tenia una gran viabilidad en aquellos momentos. Con este estudio se podían mejorar los tiempos de tratamiento empleados por los procesos de alineación en el ámbito de la biomedicina. La mejora se vio fortalecida en el momento que se identificó que los accesos a los datos podían mejorarse con la utilización de un sistema caché. El trabajo de investigación fue publicado con el nombre *"Studying the improving of data locality on distributed grid applications in bioinformatics"* [105] y se pueden ver más detalles del documento en el Capítulo 3.

El segundo reto que corresponde con la "V" de Variedad, orientó los esfuerzos hacia el mundo de las arquitecturas específicas de Big Data donde los sistemas permitían el estudio de nuevas posibilidades y trabajos de investigación en áreas más activas. Los objetivos se centraban en la realización de prototipos que permitieran demostrar que la utilización de datos multimedia era posible tanto con Apache Hadoop como con Apache Storm. De esta forma, se abarcaban dos arquitecturas de procesamiento

masivo de datos como son Lambda y Kappa. Asimismo, cada una de ellas corresponde con un período muy definido de investigación con su correspondiente publicación.

La primera parte de la investigación sobre la variedad del Big Data, permitió demostrar que la utilización de fuentes de datos multimedia es posible dentro de entornos de trabajo Apache Hadoop y bajo qué condiciones se podía realizar su tratamiento. En este caso se utilizaron fuentes de audio y herramientas específicas para su tratamiento (conversores, diarizadores, etc.) para la implementación de un caso completo de tratamiento multimedia. Como resultado se publicó la investigación con el nombre *"Detecting events in streaming multimedia with big data techniques"* [106], presentada en el Capítulo 4.

La segunda parte de esta dimensión hace que se estudien las posibles opciones de tratamiento de fuentes multimedia para la detección de logotipos mediante el uso de Apache Storm como herramienta de *stream processing*. La investigación poseía tres objetivos definidos: demostrar que no solamente se podía utilizar sonido como dato multimedia en una arquitectura de Big Data, hacerlo en una arquitectura de tipo Kappa, y por último utilizar las técnicas de *data mining* necesarias para mejorar el algoritmo utilizado. Es por ello que se aplicó el Bloom-Filter [16] como elemento de mejora en un grafo acíclico dirigido (DAG). En este caso se optó por un análisis más acorde con las líneas de investigación que se tenían establecidas en el ámbito del *Internship* realizado en el *National Institut of Informatics* de Japón. La publicación en este período tiene el nombre *"Logotype detection in streaming multimedia using Apache Storm"* [108]. El Capítulo 5 amplía la información sobre el artículo publicado.

Para cubrir todos los objetivos planteados, y para tener en cuenta la tercera "V" de Volumen, se planteó una investigación que estuviera adaptada al estado del arte y que diera respuesta a una volumétrica creciente en las arquitecturas de Big Data. Este era un caso de estudio para la flexibilidad en un entorno más amplio, como puede ser el ámbito de los contenedores software. Es por ello que se realizó un estudio para conocer la aplicabilidad de un sistema bio-inspirado en los procesos de escalabilidad de los contenedores. La posibilidad de utilizar un sistema distribuido de autoescalado que permita su utilización en una arquitectura de microservicios para el procesamiento de grandes volúmenes de datos, ha sido la inspiración para este último trabajo. La investigación demuestra que el algoritmo aplicado mejora la respuesta en caso de cargas imprevistas, siendo inferior su rendimiento en el caso de cargas predecibles. La tarea de transmisión pública del conocimiento que le corresponde a este período tiene un documento con el nombre *"Toward bio-inspired auto-scaling algorithms: An elasticity*

approach for container orchestration platforms" [107]. En el Capítulo 6 se pueden obtener más detalles.

1.4 Estructura de la tesis

Para dar una visión completa de las tareas que se describen en esta tesis, a continuación se detalla la estructura que poseen los siguientes capítulos y su intención.

En el capítulo 2 se expone brevemente la evolución histórica de los modelos de tratamiento de grandes volúmenes de datos, incidiendo especialmente en aquellas características que serán relevantes, o están relacionadas, con los documentos publicados y que permitirán una mayor comprensión de las líneas principales de investigación. De esta forma se mencionan los sistemas Grid, las arquitecturas de procesamiento de grandes volúmenes de información, Lambda y Kappa, así como las arquitecturas orientadas a microservicios.

En los capítulos 3, 4, 5 y 6 se repasan los artículos publicados, que se presentan como parte de los mismos. La descripción y justificación de las investigaciones que han llevado a la publicación son la motivación principal de estos capítulos. Cada uno de los capítulos se inicia con una referencia a la publicación y sus condiciones particulares de difusión que la habilita para su utilización en la presente tesis. Después de esta referencia, y justo antes del *Abstract* donde comienza el texto publicado, se ha optado por incluir un pequeño resumen de toda la publicación con los aspectos más destacables.

Siguiendo la secuencia, es en el capítulo 7 donde se muestran los resultados que se han obtenido. Es aquí donde se menciona el software desarrollado, las conferencias donde se han presentado y las colaboraciones gracias a las que se han podido progresar.

Es en el último capítulo 8 donde se exponen las conclusiones finales, líneas futuras de investigación y fondos económicos que han permitido realizar los estudios. Es en este capítulo donde se analiza la aplicabilidad de los conocimientos adquiridos en los procesos de investigación de la tesis.

En las secciones finales se encuentra la bibliografía y dos apéndices, el primero con los permisos necesarios para realizar la publicación de los documentos que contiene esta tesis, y la segunda con información adicional sobre los congresos y revistas donde se han realizado, o bien comunicaciones orales, o bien publicaciones relacionadas con esta tesis.

Capítulo 2

Descripción tecnológica del procesamiento automatizado de datos masivos

Los tratamientos que requieren una elevada capacidad de procesamiento o de recursos han sido siempre una de las mayores preocupaciones dentro del ámbito de la computación. Durante muchos años estas capacidades se obtenían y gestionaban de forma centralizada sobre sistemas de elevadas prestaciones. Con la aparición de los sistemas de menor coste y el aumento de la capacidad de comunicación entre los diferentes nodos, se diseñaron arquitecturas que unían tanto las altas capacidades como los reducidos costes.

Los sistemas que se están utilizando para el tratamiento de grandes volúmenes de datos en las últimas décadas han evolucionado [154] junto a las necesidades que los propician, dando lugar a un elevado número de alternativas disponibles en la actualidad.

A más alto nivel podemos diferenciar claramente dos tipos de procesamiento de datos [218]. El modo *batch* que permite procesar grandes volúmenes de datos en tiempos espaciados y el modo *stream*, también conocido como tiempo real o semi-real, que permite procesar datos casi en el mismo instante de su producción.

Cuando se trata de grandes conjuntos de datos se hace referencia al orden de petabytes o zettabytes que tienen como fuente procesos tecnológicos como meteorología, genómica, conectómica, simulación de procesos físicos, investigaciones relacionadas con procesos biológicos o ambientales, motores de búsqueda en Internet, sistemas financieros o de negocios. Pero podemos referirnos también a aquellos datos que tienen como fuentes los sensores, las actividades relacionadas con dispositivos móviles, las bitácoras de funcionamiento, las cámaras en sistemas de teledetección, micrófonos, etc.

Con el objetivo de poder destacar aquellas características arquitectónicas importantes en los sistemas de procesamiento, en las próximas secciones se detallan brevemente aquellas relevantes y que resultan de interés para su mención posterior en el presente documento.

2.1 Arquitectura de infraestructuras Grid

El modelo Grid aparece como respuesta a la necesidad de integrar sistemas distribuidos, heterogéneos a gran escala, combinando recursos geográficamente distribuidos con dominios de administración diferenciados. Se diseñaron como complemento de los sistemas locales de gestión de recursos (LRMS – Local Resource Management System) que ofrecen una visión uniforme y única de los equipos, una gestión de trabajos optimizando el uso, asumiendo un sistema de colas para servidores HPC (High Performance Computing), una gestión de recursos en clusters dedicados y un gestor de la carga para los sistemas hardware. Los LRMS tienen ciertas desventajas debido a que no disponen de interfaces ni infraestructuras comunes, sino que se basan en protocolos propietarios, creando silos computacionales incompatibles incluso en una misma organización. Esto implica que los recursos solo están disponibles para un grupo muy reducido de usuarios finales. Si introducimos el Grid obtenemos acceso transparente a los recursos subyacentes [114] [223].

A esta problemática debemos añadir aquellos casos en los que el procesamiento masivo de datos excede fácilmente las capacidades de procesamiento de los ordenadores convencionales. Por ello no es extraño encontrar procesos de tratamiento en los que los datos procesados superan las altas capacidades de almacenamiento y procesamiento de los ordenadores operativos en la organización. Asimismo, hay que tener en cuenta que no se desea aumentar significativamente el tiempo de procesamiento ni tampoco incrementar considerablemente los gastos operativos de la infraestructura.

Con vistas a ampliar el campo de uso de estos sistemas aparecen los *middleware Grid* que permiten compartir recursos de naturaleza dispar (software, hardware y datos) entre entidades de forma dinámica, creando una arquitectura de protocolos comunes e integrados. De esta forma se permite a las organizaciones colaborar entre sí, compartiendo recursos con la finalidad de alcanzar objetivos compartidos. Dentro del nuevo nivel de abstracción se crean interfaces e infraestructuras colectivas que están coordinadas y no se encuentran sujetas a un control centralizado.

La principal tarea de un Grid es integrar y compartir los recursos de diferentes proveedores entre usuarios por medio de una WAN (Wide Area Network) [65]. Es

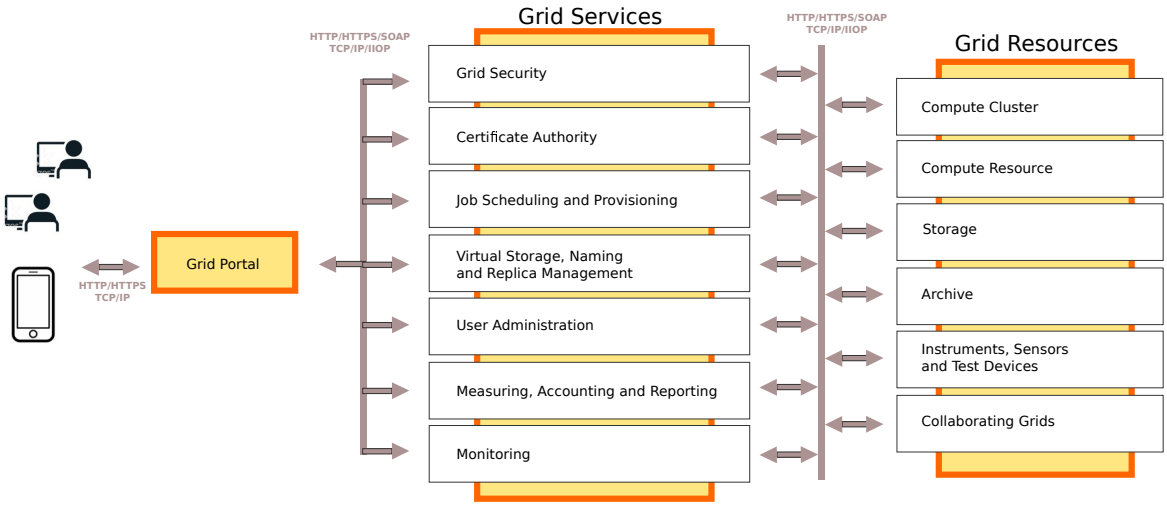


Fig. 2.1 Vista lógica de un Grid.

común encontrar descrita la arquitectura del Grid en términos de capas, asignándole a cada una de ellas una funcionalidad específica.

- **Capa de aplicación.** La compone todo el software de aplicaciones de usuario y herramientas de desarrollo que utilizan las mismas. Es la capa que ve el usuario y por ello recibe el nombre de "serviceware".
- **Capa de middleware.** Es la capa que permite que los recursos participen de forma coordinada y segura en el Grid.
- **Capa de recursos.** Es la compuesta por las capacidades físicas disponibles en el Grid: ordenadores, supercomputadores, sistemas de almacenamiento, catálogos electrónicos de datos, bases de datos, sensores, etc.
- **Capa de red.** Responsable de organizar las conexiones entre los recursos individuales de la capa anterior.

La responsabilidad del desarrollo de estándares recae en la OGF (Open Grid Forum) [63], comunidad de usuarios, desarrolladores y vendedores que aúnan esfuerzos en la estandarización mundial del *Grid computing*. La arquitectura basada en *Web Services* es conocida como OGSA (Open Grid Services Architecture) [67] [64] y aparece como la referencia clave para los proyectos Grid. La principal implementación de OGSA es Globus Toolkit [9] (actualmente retirado y solamente disponible como freeware), infraestructura de código abierto desarrollada por la Globus Alliance [8] que proporciona herramientas de programación Java (librerías, servicios y APIs) para el desarrollo de

aplicaciones Grid basándolas en sus servicios y capacidades (seguridad, localización, gestión de recursos, etc).

EGI (European Grid Infraestructure) o EGI.eu [37] es una organización sin ánimo de lucro que proporciona servicios de cómputo avanzado para la investigación e innovación. Su infraestructura está compuesta por centenares de centros de proceso de datos y proveedores de servicios en la nube distribuidos a lo largo del mundo. Los servicios suministrados se centran actualmente en proporcionar servicios de computación de altas prestaciones y *Cloud services*. Uno de los miembros de está estructura europea es IberGrid (Iberian Grid Infrastructure) [125], organización que esta compuesta por la INCD portuguesa [43] y la ES-NGI de España, ambas parte de la *National Grid Initiatives* (NGIs).

El software gLite [195] (retirado desde mayo de 2013) era el *middleware* que daba soporte a los experimentos del CERN LHC [194]. Basado en Globus Toolkit poseía una librería, GFAL [196], implementada en C que proporciona una capa de abstracción para el sistema de almacenamiento del Grid y el acceso a ficheros locales. Actualmente, se utiliza la UMD (Unified Middleware Distribution) [41] para soportar la infraestrutura Grid en EGI.

Aunque las clasificaciones del Grid pueden ser variadas, en función de los criterios utilizados podemos realizar agrupaciones en base a la funcionalidad:

- **Grid computacional** (*Computational Grid*), que hace referencia a la potencia del proceso agregado procedente del uso de un conjunto de sistemas distribuidos.
- **Grid de Datos** (*Data Grid*), que hace referencia a la existencia de una arquitectura de datos que se integra de forma coordinada para gestionar, como si de un único recurso se tratara, los datos que aparecen de diversas aplicaciones intensivas en datos.
- **Grid de Colaboración** (*Collaborative Grid*), que se centra en la capacidad de un potente entorno donde pueden interactuar diferentes agentes dispersos geográficamente que colaboran en un trabajo.
- **Grid de Servicios** (*Utility Grid*), se refiere a una plataforma amplia de servicios, no sólo de computación o de almacenamiento. Es la integración que proporciona toda la potencia de sus recursos a dispositivos ligeros y de poca potencia.

Aquí es donde podemos comenzar a introducir el término *Data Grid* como el conjunto de servicios necesarios para el procesamiento, transmisión y almacenamiento

de grandes volúmenes de información, haciendo disponibles estos sistemas a la totalidad de usuarios/colaboradores de la infraestructura.

Durante muchos años la tecnología Grid ha ayudado a los científicos a tratar grandes volúmenes de datos haciendo uso de las características de la infraestructura que permiten el almacenamiento de datos distribuido, la instrumentación remota, las colaboraciones complejas y los trabajos de computación distribuidos. Es así como se han alcanzado unos niveles de consenso adecuados para llevar a cabo investigaciones que, sin una infraestructura adecuada, habrían sido imposibles. Es el caso de la confirmación de la existencia del bosón de Higgs por el CERN-LHC [194] permitiendo que el análisis de los datos obtenidos de los experimentos ATLAS [192] y CMS [193] se procesaran a una velocidad no obtenida hasta el momento, en un volumen de datos tan grande como el que ha sido necesario para obtener la certeza de su existencia.

Haciendo uso de los estándares SOAP (Service Object Access Protocol) [17] y WSDL (Web Service Description Language) [30] se ha construido una arquitectura basada en servicios SOA (Service Oriented Architecture) [165] que normaliza la sintaxis para la descripción de los servicios y los protocolos de comunicación de los mensajes necesarios para interactuar entre los diferentes componentes de la infraestructura.

Como todos los sistemas Grid tienen algunas desventajas que hacen que no sean recomendables en algunos casos. En ocasiones, las aplicaciones no se pueden dividir en tareas más pequeñas. Si esta división no se realiza, los beneficios de la ejecución en Grid no aparecen. Las aplicaciones monolíticas necesitan un proceso de adaptación para obtener beneficios con la ejecución en Grid. En ocasiones, la ejecución de aplicaciones es muy complicada considerando las interacciones con usuarios, sistemas, bases de datos, almacenamiento, etc. Al final, un Grid es un conjunto integrado de *hosts* y, por tanto, se deben tener en cuenta también los problemas organizativos que supone el mantener la infraestructura. Los modelos de explotación y costes, las políticas de seguridad y los dominios de administración son tareas que hay que manejar correctamente para disponer de un Grid útil. Por último, en algunos casos, la comunicación es lenta y no uniforme, derivada de la distribución geográfica, lo que supone dificultades en la sincronización de los procesos.

Los posibles ejemplos de la utilización de Grid para el tratamiento de grandes volúmenes de datos lo podemos ver en [91] o [142]. En ambos casos se presentan soluciones donde se combina un Grid con los sistemas de almacenamiento propios de los sistemas de Big Data.

2.2 Arquitecturas para el procesamiento masivo de datos

El procesamiento *batch* fue durante muchos años la única alternativa válida para el procesamiento de grandes volúmenes de información. A las soluciones implementadas en Grid se les fueron sumando las basadas en MapReduce [45] como un intento de situar los datos en el nodo de procesamiento para que su tratamiento sea siempre local. Esta característica de localidad de los datos reconoce que el bien más preciado es el ancho de banda de las comunicaciones. A este panorama se le agregó la necesidad de un tratamiento más rápido para el procesamiento en *streaming*. La mezcla de ambas, *batch* y *streaming*, resultó ser la propuesta arquitectónica inicial para una solución completa y compleja denominada arquitectura Lambda [139]. Los problemas que aparecen con el mantenimiento de dos paradigmas operando al mismo tiempo hacen que la arquitectura evolucione a una alternativa más mantenible y ligera, la arquitectura Kappa [186].

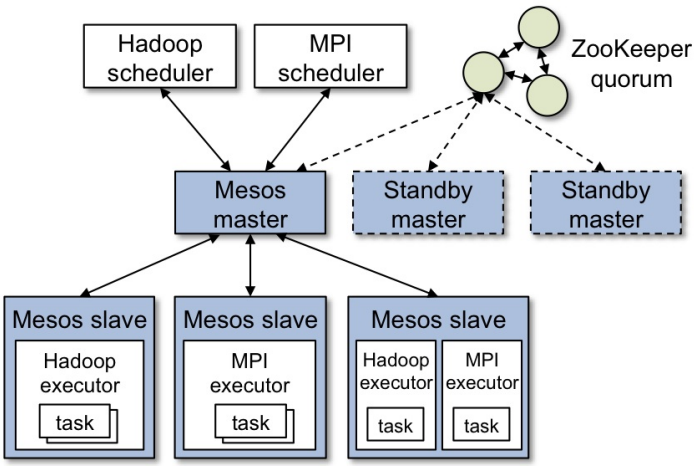


Fig. 2.2 Arquitectura Apache Mesos [23].

Aunque existen propuestas de arquitecturas basadas directamente en productos como Apache Mesos [71] (ver Figura 2.2) o Apache Hadoop YARN [232] (ver Figura 2.3), en esta sección trataremos las alternativas de arquitecturas de procesamiento de datos independientemente de los productos tecnológicos que existan en cada momento. Es cierto que en ocasiones las arquitecturas descritas se han desarrollado en base a un producto innovador, pero hay que verlas como un contenedor de componentes de trabajo que se pueden encajar dentro de un esquema global de colaboración con otros productos tecnológicos.

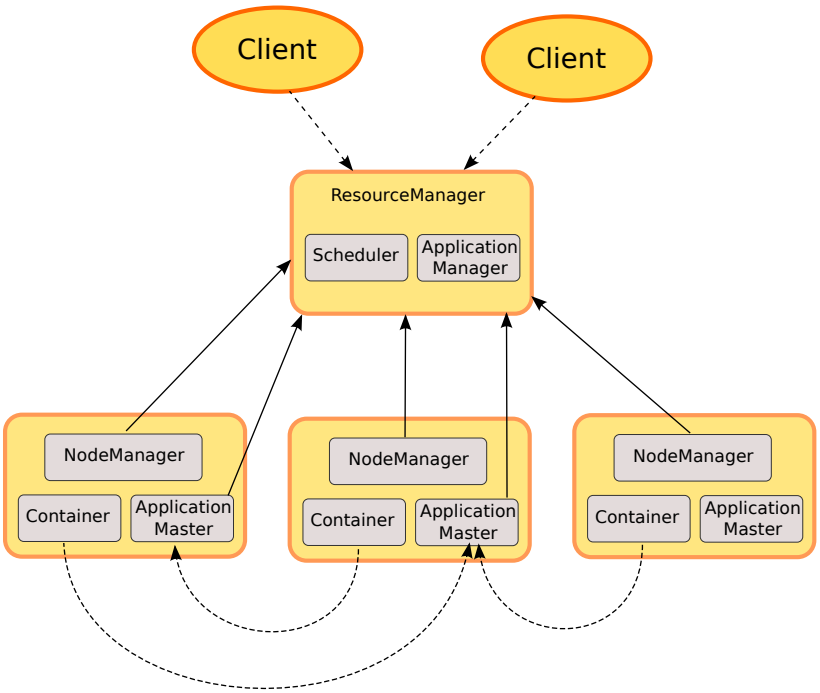


Fig. 2.3 Arquitectura Apache Hadoop YARN (extraído de [70]).

En esta sección, se describirán brevemente las características de las dos principales arquitecturas de Big Data (Lambda y Kappa).

2.2.1 Arquitectura Lambda (Λ)

La arquitectura de procesamiento de datos Lambda [171] aparece en el año 2012. Atribuida a Nathan Marz, este la definió en base a su experiencia en los sistemas de tratamiento de datos distribuidos [170]. La idea principal desarrollada es la combinación de las dos modalidades de procesamiento: *batch* y *streaming*, respetando en cada momento el sistema de tratamiento más adecuado para trabajar con el Big Data. Esto nos proporciona lo mejor de los dos mundos, con alcance completo para el modo *batch* y decisiones semi-instantáneas en el modo *streaming*.

Por un lado los sistemas *batch* manejan correctamente los errores que se producen en el procesamiento de los datos, mientras que los sistemas de *streaming* deben garantizar la idempotencia, como la capacidad de realizar una acción un número indeterminado de veces y obtener el mismo resultado que si se realizara solo una vez.

Como se muestra en la Figura 2.4 esta arquitectura está compuesta de tres capas:

- La **capa por lotes**, *Batch Layer* o *Cold Path*. Esta capa tiene como función principal la gestión del repositorio principal de datos en bruto (*raw data*). Este

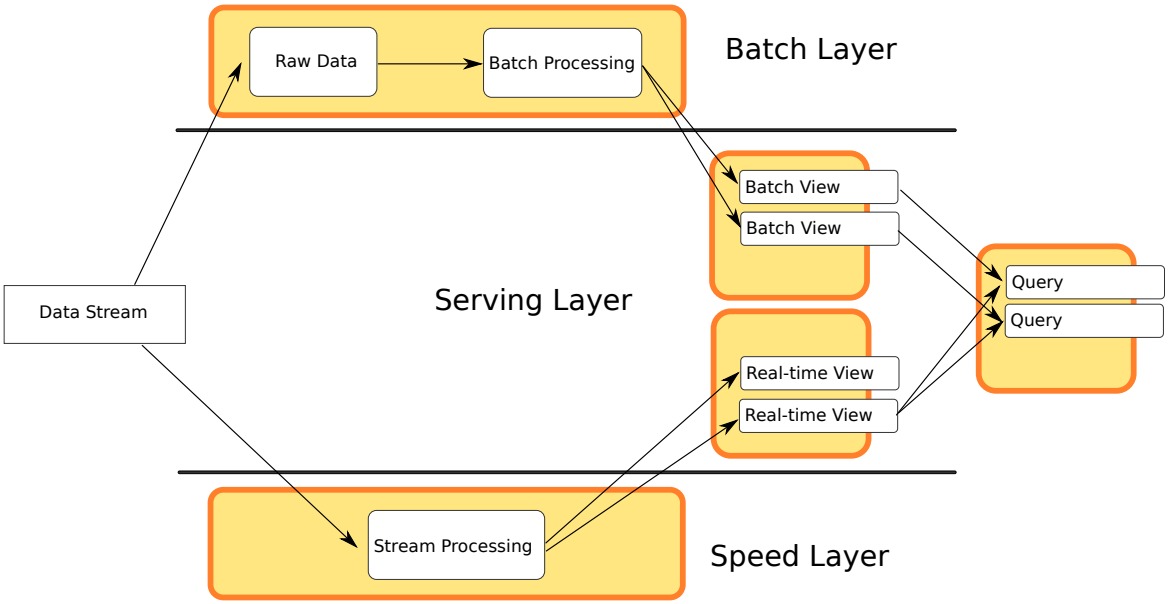


Fig. 2.4 Diagrama de capas. Arquitectura Lambda (basado en [147]).

almacén, que es inmutable, solo admite procesos incrementales de inclusión de datos originales. Como segunda función se destaca la tarea de preprocesar los datos para la capa de servicios en su parte de vistas *batch*.

- La **capa de velocidad**, *Speed Layer* o *Hot Path*, es la responsable de efectuar el cómputo en tiempo real. Las vistas realizadas en tiempo real son transitorias ya que, tan pronto como los datos se propaguen a la capa *batch* y de servicio, éstas dejarán de ser válidas. Es en esta capa donde se pueden aplicar algoritmos de Machine Learning.

- La **capa de servicios** o *Serving Layer*. Permite el acceso rápido y avanzado a los resultados del procesamiento.

Las tecnologías utilizadas en estas capas son variadas y en ocasiones compartidas. Tomando como base las tres capas podemos mencionar: para la capa *batch*, Apache Sqoop [82] como herramienta para la transferencia de datos, HDFS [76] y Cassandra [73] como tecnologías de almacenamiento o Hadoop [70], Pig [79] y Hive [77] para el procesado; para la capa de velocidad, Apache Kafka [78] como manejador de streams, Apache HBase [75] como almacenamiento y Apache Spark [81], Apache Storm [83] o Apache Samza [80] para procesamiento; en la capa de servicios encontramos almacenes NoSQL como Apache HBase, Cassandra, MongoDB [124] o Neo4j [227]. En [95] existe una clasificación bastante exhaustiva de herramientas disponibles.

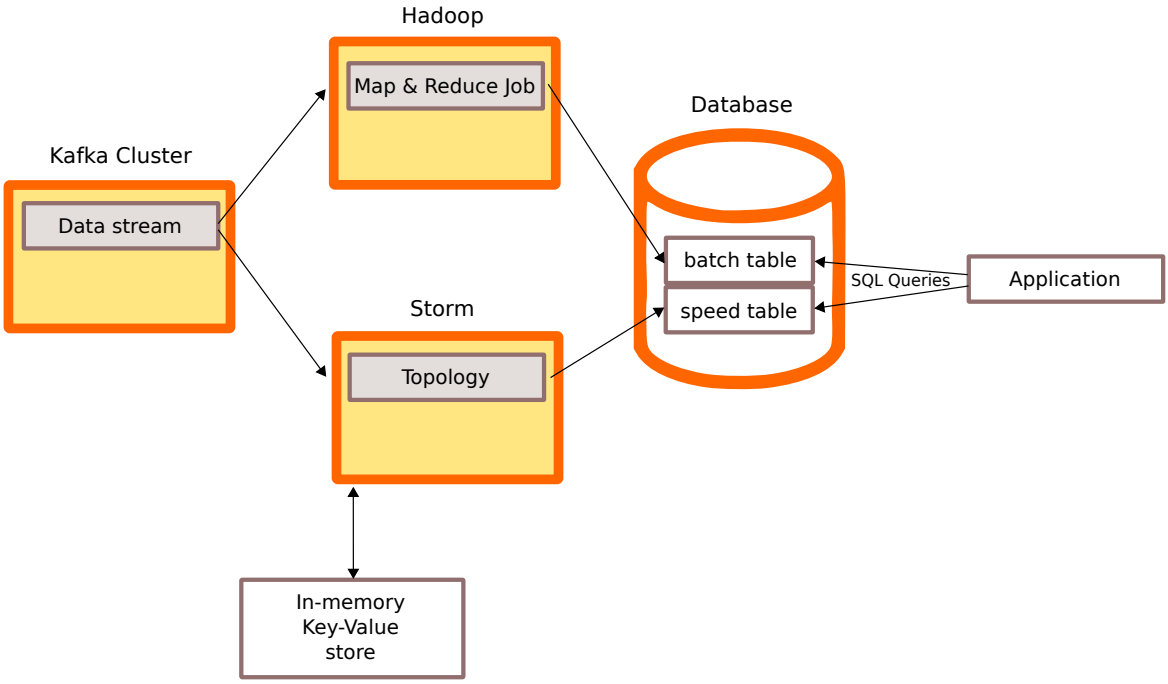


Fig. 2.5 Diagrama ejemplo implementación arquitectura Lambda.

Uno de los objetivos principales de este tipo de arquitectura es la integración del *batch processing* y el *stream processing*. De hecho, un gran número de artículos sobre esta arquitectura se centran en la capacidad de reunir tipos de procesamiento diferentes [145] [139]. Aunque como contrapartida tiene sus problemas: la no existencia de procesamiento de transacciones o las pobres posibilidades de acelerar los procesos que no pueden ser paralelizados con éxito; no se pretende la utilización de datos con requisitos de baja latencia; no están diseñados para procesar muchos ficheros pequeños, y aunque lo puede hacer, no en tiempo real.

En cualquier caso se deben tener en cuenta varias características:

- **Complejidad.** La existencia de muchas capas por donde fluyen los datos dificulta mantener la sincronización durante la operación del sistema.
- **Mantenimiento y soporte.** Ya que está definido por dos pilas claramente diferenciadas (velocidad y *batch*) el soporte y mantenimiento es difícil.
- **Tecnología.** La aplicación de numerosas tecnologías expone al sistema a nuevas incertidumbres. La inclusión de tecnologías emergentes que mejoran el procesamiento, almacenamiento o la interacción entre componentes supone siempre un riesgo.

- **Implementación y despliegue complejos.** Poner en práctica una arquitectura Lambda utilizando tecnologías *open-source* o utilizarla vía Cloud *on-premises* puede ser una tarea compleja, especialmente si lo comparamos con la utilización de un único producto consolidado y sin integración con otros.

Es común ver esta arquitectura implementada en organizaciones como resultado de la evolución de los sistemas hacia las necesidades de operación. De esta forma, a un sistema totalmente *batch* se le ha incorporado un sistema de *streaming* para solventar problemas existentes. Es común utilizar una base de procesamiento *batch* común en sistemas de *Business Intelligence* y añadirle una capa de velocidad para solventar la operación de *streaming*. En este grupo se encontrarían las recomendaciones para el uso de implementaciones como la arquitectura unificada de Oracle [35] o IBM Big Data Platform [216] (ver Figura 2.6).

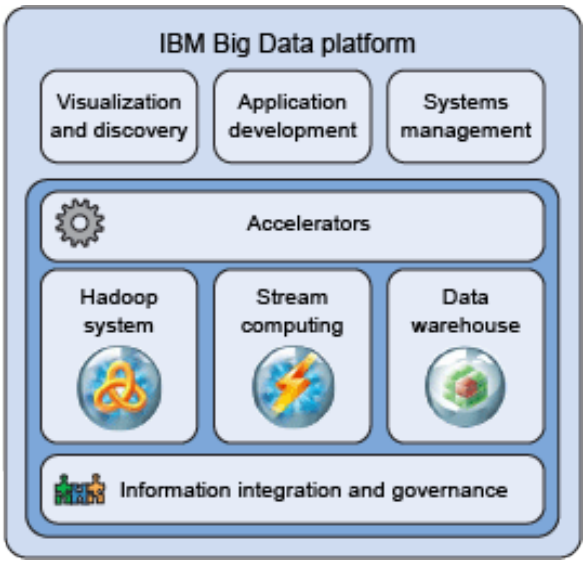


Fig. 2.6 Arquitectura de IBM para Big Data [59].

Algunas de las posibilidades existentes para la implementación de la capa de *streaming* se muestran en la Tabla 2.1, donde podemos ver el nombre de la tecnología utilizada, el modelo computacional, si trata tuplas como unidad mínima o bien es *micro-batch* (pequeños procesos *batch*), si almacena el estado entre un instante de ejecución y el del siguiente (*stateful*) o bien no conserva esta información (*stateless*) y como última columna se incluye información sobre la gestión de la tolerancia a fallos que soporta.

Nombre	Modelo computacional	Gestión del estado	Tolerancia a fallos
Spark Streaming	Micro-batch	Stateful	Exactly-once
Storm	Tuplas	Stateless	At-least-once
Trident	Micro-batch	Stateful	Exactly-once
Heron	Tuplas	Stateless	Exactly-once At-most-once At-least-once
Flink	Tuplas	Stateful	Exactly-once
Samza	Tuplas	Stateful	At-least-once
MilWheel	Tuplas	Stateful	Exactly-once
S-Store	Tuplas	Stateful	Exactly-once

Tabla 2.1 Comparativa de motores de procesamiento en *streaming* [225][213][224].

2.2.2 Arquitectura Kappa (K)

La arquitectura Kappa [144], mostrada en la Figura 2.7 y clasificada como una arquitectura de mensajes distribuida, fue introducida por Jay Kreps en 2014. Kreps es también el creador de Apache Kafka y Voldemort (*key-value store*). Era ingeniero en LinkedIn además de ser fundador y CEO de Confluent, Inc. empresa del sector del Big Data. Esta arquitectura es una versión simplificada de la arquitectura de procesamiento de datos Lambda sin la opción de procesamiento *batch*. No representa una arquitectura que sustituya a otra sino que se considera como una alternativa donde el procesamiento por lotes no se hace necesario.

La idea principal de la arquitectura Kappa es lograr un procesamiento *real-time* tratando constantemente la información a través de un único esquema de procesamiento de *streams*. De esta forma logra simplicidad comparada con la arquitectura Lambda, manteniendo un solo código en vez de manejar dos códigos correspondientes a la parte *batch* y al procesamiento *streaming*.

Kreps propone un flujo de datos inmutable que contiene toda la información que se desea persistir indefinidamente o para un período de tiempo determinado. De esta manera se permiten pruebas unitarias y revisiones de los cálculos realizados que no se permiten en Lambda. La inmutabilidad de los datos se consigue mediante los registros

de la propia herramienta de procesamiento. En la práctica, este almacenamiento se realiza, por ejemplo, mediante el registro de acontecimientos de Apache Kafka o un *key-value store*. Debemos tener en cuenta que, en la arquitectura Lambda, la persistencia se consigue mediante tecnologías como Hadoop/HDFS.

Los procesadores de *streams* tratan todos los datos de las fuentes y los guardan en un soporte definitivo como una tabla (capa de servicios). Si se necesita otro reprocesamiento se toman los datos originales almacenados sin modificar y se vuelven a tratar. Luego se pueden guardar en un segundo almacén. Cuando se termina el segundo procesamiento se cambia el origen de los datos de las aplicaciones del primero, que contenía el resultado del primer tratamiento, por el segundo repositorio ya actualizado.

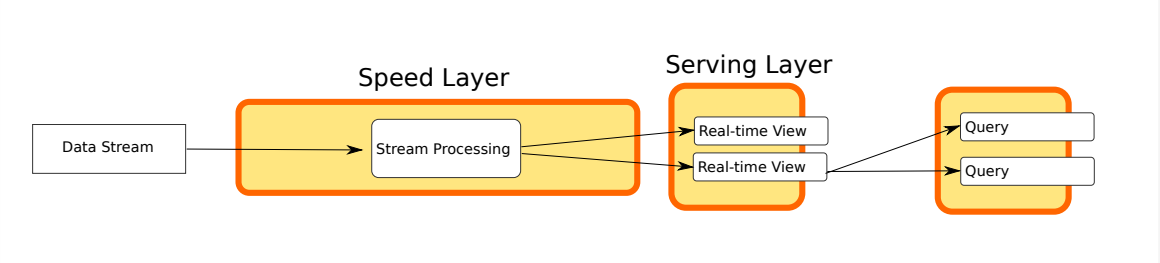


Fig. 2.7 Diagrama de capas. Arquitectura Kappa.

Utilizar *workflows* o *pipelines* es uno de los retos que se plantea un arquitecto de soluciones en el momento de comenzar el diseño de una arquitectura de tratamiento de datos. La principal diferencia entre el *workflow* tradicional y el *Big Data pipeline* puede centrarse en que se permite escalar todos los datos a un *data lake* que es un repositorio común para toda la organización. Son tecnologías no relacionales (NoSQL) las cuales no requieren un esquema predefinido sino que se crean en función de las necesidades existentes en cada momento. Hay dos tipos de procesamiento: tiempo real (*"in-memory"*) y sobre flujo continuo de datos (*"streaming"*). Existen multitud de herramientas disponibles entre las que podemos nombrar Apache Storm.

En ciertas ocasiones, las estrategias de funcionamiento se basan en tener un sistema de mensajería (por ejemplo Apache Kafka) que mantenga un registro de los datos a procesar. En el momento que se necesita reprocesar se utiliza este almacén permanente.

Uno de los posibles ejemplos de su utilización, en un entorno de producción, está en la Figura 2.8 donde se describe la operación en tiempo real de los datos procedentes de los entornos web y de dispositivos móviles en la empresa Facebook, Inc.. En la Figura 2.9 se puede observar otro ejemplo, esta vez de Pinterest, en el que se puede apreciar como Apache Kafka es el eje central del tratamiento.

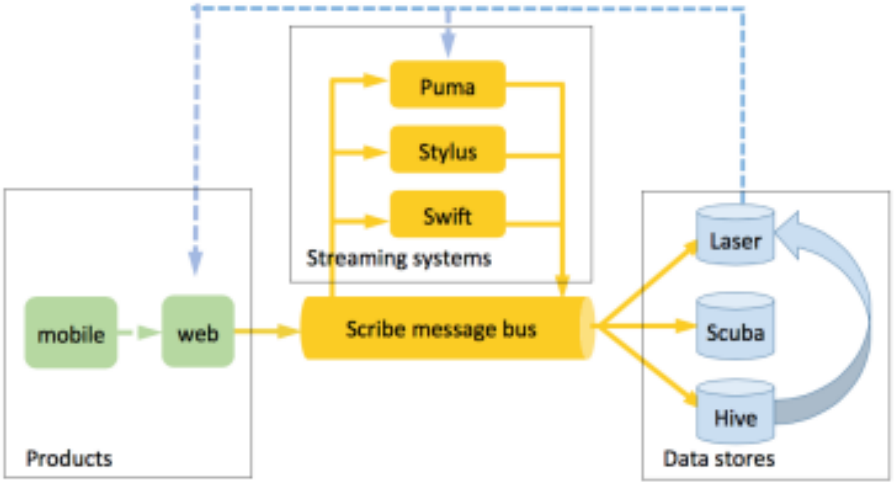


Fig. 2.8 Ejemplo procesamiento arquitectura Lambda en Facebook [28].

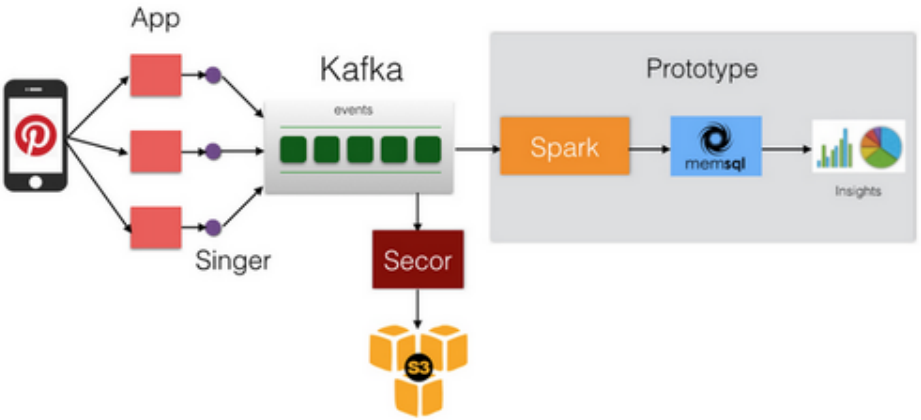


Fig. 2.9 Ejemplo procesamiento arquitectura Lambda en Pinterest [87].

Algunas características de estos sistemas se pueden definir en función de los datos que se pretenden tratar. Hay un grupo elevado de sistemas en los que la tasa de llegada de datos no está bajo el control del sistema. En general, esta tasa de recepción de datos es mayor que la capacidad de procesamiento que tiene el sistema, por lo que es necesario aplicar mecanismos para aumentar la capacidad, así como reducir los recursos en los momentos adecuados. Otra posibilidad que se plantea a los diseñadores está relacionada con el orden de llegada de los datos, que no se garantiza. Algunos datos pueden retrasarse respecto a otros que se han generado posteriormente. Durante el procesamiento, debe asumirse que los datos son imperfectos y que debe ajustarse la respuesta ante estas imperfecciones. La única solución no es descartarlos, sino que se debe encontrar el adecuado equilibrio entre el procesamiento en tiempo real y una búsqueda de la corrección completa. Si tenemos en cuenta que estamos procesando datos que aparecen en un *stream* debemos plantear la posibilidad de que evolucionen en el tiempo. Este tratamiento de las series temporales de datos debe tenerse perfectamente definido en el código.

De hecho podemos destacar algunas características fundamentales que debemos manejar en sistemas de *streaming*:

- Tolerancia a fallos. Los procesos deben tener en cuenta que se pueden producir problemas durante el proceso de tratamiento y se deben tomar medidas para que el sistema permanezca en un estado predecible y conocido.
- Pérdida de mensajes. Durante el movimiento de grandes volúmenes de información se debe asumir que, si es posible que se tengan pérdidas de datos, estas pérdidas se producirán tarde o temprano. Por ello, hay que preparar sistemas para que esta contingencia resulte minimizada.
- Garantía de entrega de mensajes (ACK). Se debe poder asegurar que los procesos confirman la recepción y tratamiento de un dato determinado. Por ello podemos tener diferentes tipos (ver Tabla 2.1 para obtener ejemplos):
  - *At-least-once*. Se entrega el mensaje al menos una vez.
  - *At-most-once*. Se entrega el mensaje como máximo una vez, pero puede no entregarse.
  - *Exactly-once*. Se entrega el mensaje exactamente una sola vez.
- Estabilidad. Se puede suponer desde un principio que tendremos problemas en los nodos debido a caídas. Debemos tener en cuenta la habilidad del sistema para

procesar los datos de forma redundante o que se pueda recuperar procesando de nuevo aquellos no tratados. Para una estrategia de recuperación podemos disponer de *Checkpoints/logs* o *buffers* para retransmisión.

La implementación de esta arquitectura se ha realizado en múltiples variantes de aplicaciones de redes sociales, en sistemas de monitorización de dispositivos conectados a un Cloud o en aplicaciones de Internet de las Cosas (IoT). Hay algunas ocasiones en las que las arquitecturas Kappa son recomendables [169][210]:

- Si se precisa de un sistema altamente disponible con una elevada tasa de tratamiento de datos y volumen de datos del orden de Terabytes para cada nodo, además de admitir replicación en el procesamiento.
- Aquellos sistemas que mantienen un alto volumen de usuarios online, como pueden ser los sistemas de recomendaciones, siempre que no sea precisa la utilización de sistemas *batch*.
- En aquellas ocasiones en los que el retratamiento solo es necesario por el cambio de código.
- Si los procesos son escalables horizontalmente de manera que pueden ser descompuestos en problemas más pequeños que se procesan de forma independiente y paralela.
- En aquellos casos en donde los tratamientos no son computacionalmente exigentes y poseen un número elevado de datos.

Por contra, la ausencia de la capa *batch* hace que la detección de los errores en el procesamiento sea más compleja. De hecho, se hace necesario disponer de un manejador de excepciones para poder tratar aquellos datos que han tenido problemas en el primer procesado y después poder reconciliar estos datos procesándolos posteriormente.

2.3 Arquitecturas basadas en microservicios

Segun Lewis and Fowler [157], el término *"microService"* fue acuñado en un *workshop* de arquitectura del software celebrado cerca de Venecia en el año 2011. En esta reunión, varios participantes defendieron una misma idea compartida, una forma de arquitectura común que se materializaría un año después. En marzo de 2012 en la *33rd Degree Conference for Java Masters* en Cracovia, James Lewis (en *"Microservices - Java,*

the Unix Way" [156]) y, al mismo tiempo, Fred George (en *"μService Architecture: A Personal Journey of Discovery"* [93]) expusieron lo que al final se denominaría arquitectura de microservicios.

Para el término "microservicio" tenemos varias definiciones: la concisa de Newman que remarca su independencia en [183]: *"Microservices are small, autonomous services that work together"*; la breve de Dragoni et al. que resalta su forma de relacionarse en [50]: *"A microservice is a cohesive, independent process interacting via messages"* o la más extensa de Nadareishvili et al. en [180]: *"A microservice is an independently deployable component of bounded scope that supports interoperability through message-based communication. Microservice architecture is a style of engineering highly automated, evolvable software systems made up of capability-aligned microservices"*. También podemos obtener la definición de arquitectura de microservicios de Dragoni et al. [50]: *"A microservice architecture is a distributed application where all its modules are microservices"*.

Uno de los elementos más importantes en una arquitectura es el nivel de granularidad y su accesibilidad a través de las API. Los microservicios deben diseñarse, implementarse y escalarse de manera independiente, permitiendo una entrega más rápida de las funcionalidades que contienen con un impacto lo mínimo posible. Por lo tanto, el desafío es abordar un nivel del dominio lo suficientemente concreto para evitar la orquestación de "nanoservicios", pero lo suficientemente amplio para limitar las dependencias necesarias (ver Figura 2.10).

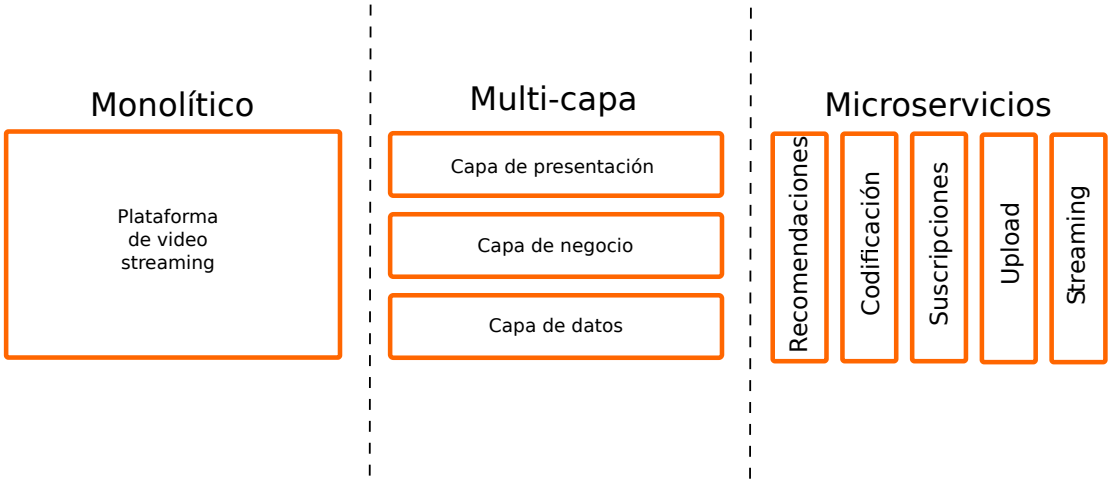


Fig. 2.10 Ejemplo de división en microservicios de una aplicación monolítica o multi-capa.

Entre los muchos atributos que poseen los microservicios, se pueden destacar los siguientes:

- **Interoperabilidad.** Ya que pueden ser implementados en varias plataformas, uno de los puntos críticos es su capacidad para intercambiar información. Esta capacidad de intercambio de datos se realiza mediante RESTful API [202]. Con el único conocimiento de este punto de intercambio de información se puede obtener el servicio en su totalidad.

- **Fiabilidad.** Definida como la capacidad del software de mantenerse operativo y funcional en condiciones normales. Ya que los componentes son independientes y aislados, la fiabilidad del sistema es mayor. Es más sencillo solucionar un fallo en un microservicio sin afectar al resto del sistema.

- **Disponibilidad.** La puesta en producción de nuevas funcionalidades del software desarrollado como microservicios requiere menos tiempo. De esta forma se favorece la disponibilidad al reducir los tiempos de inactividad.

- **Tolerancia a fallos.** Es decir, la propiedad de un sistema de seguir funcionando correctamente en caso de fallo de uno o varios de sus componentes. Los microservicios están diseñados para funcionar en sistemas distribuidos altamente fiables. Los riesgos de fallos se deben solucionar por medio de técnicas como la replicación y la distribución.

- **Flexibilidad.** Es la habilidad del sistema para adaptarse a los cambios, incluyendo la facilidad de reconfiguración y adaptación a nuevos requerimientos. Debido a la división de la lógica de negocio en componentes o servicios que son independientes, se adquiere una mayor flexibilidad a cambios.

- **Mantenibilidad.** Capacidad de un sistema para permitir el cambio de sus componentes con el fin de corregir errores, cambiar funcionalidades y/o satisfacer nuevos requerimientos a nivel de lógica de negocio. El desacoplamiento entre componentes que promueve la arquitectura de microservicios permite modificar un servicio sin alterar el resto de la infraestructura.

- **Portabilidad.** La capacidad del sistema para ser transferido a otro entorno y coexistir con otras aplicaciones compartiendo recursos del sistema. Como los microservicios están empaquetados de forma estandarizada, permiten introducir las dependencias y que se ejecuten del mismo modo en cualquier ambiente.

- **Escalabilidad.** La capacidad para crecer sin aumentar su complejidad ni disminuir su rendimiento. Tal vez sea la característica más nombrada de esta arquitectura, la capacidad de desplegar y escalar mediante la variación del número

Service-Oriented Architecture	Microservices
Autocontenidos, servicios monolíticos	Pequeños, descompuestos, aislados y desplegables independientemente
Las comunicaciones se realizan por medio de un ESB (Enterprise Service Bus)	Las comunicaciones se realizan por medio de protocolos e interfaces ligeras y estándares
Stateful, necesitando tener muy en cuenta las dependencias entre servicios cuando se introducen cambios	*Stateless*, menos sensibles a los cambios introducidos
Tiempos de inicialización prolongados	Rápido inicio y parada de servicios
Construcción alrededor de los servicios	Construcción alrededor de las capacidades

Tabla 2.2 Comparación de arquitecturas SOA y microservicios [133].

de instancias de contenedores o bien modificando sus características. Se logra una alta escalabilidad horizontal para atender la demanda creciente/decreciente, siendo solo necesario dimensionar el servicio y no el resto de la aplicación.

En ocasiones se considera a los microservicios como una parte de la arquitectura SOA (Service Oriented Architecture) con la diferencia que la arquitectura de microservicios a menudo utiliza *DevOps* como metodología para crear un sistema de despliegue continuo. Pero hay que tener en cuenta que existen ciertas características y diferencias que separan ambas visiones como se muestra en la Tabla 2.2.

Los contenedores software [178] son la plataforma ideal para desarrollar microservicios, pero no son apropiados para todos los escenarios. Los microservicios *stateless* no mantienen la información de estado entre llamadas, pueden recibir, procesar y enviar respuestas a peticiones sin persistir ninguna información de estado entre llamadas. Un microservicio *stateful* persiste el estado entre diferentes llamadas, viéndose obligado a guardar permanentemente la información. En cualquier caso, este estado nunca se almacenará de forma local sino que lo persistirá de forma externa en algún tipo de almacén de datos como puede ser un sistema de base de datos relacional, NoSQL o cualquier otro almacén con duración indefinida.

2.3.1 Escalabilidad en arquitecturas de microservicios

Escalamos los servicios por dos razones principales. La primera razón que nos lleva al escalado es para dar respuesta a los fallos. En el caso que se produzca un fallo en un sistema podemos tener otro u otros preparados para su puesta en marcha o listos para que puedan asumir correctamente la carga que les llegará con la parada de algún otro nodo. La segunda es para solventar un problema de rendimiento. La capacidad que tiene un sistema para resolver problemas puede llegar a ser proporcional a la cantidad de elementos que tiene disponibles para el tratamiento o almacenamiento.

Por las características expuestas anteriormente, no es una coincidencia que los microservicios y el Big Data ganen popularidad al mismo tiempo. Ambos emplean aproximaciones similares para manejar los datos. En el caso del Big Data, de la manera que son procesados en Hadoop, los datos se dividen en pequeños subconjuntos que son procesados en los nodos más cercanos a los datos. Esto crea un sistema altamente escalable que es una de las características más destacadas de las arquitecturas basadas en microservicios.

Se puede inferir cómo los microservicios escalan de una forma similar. Su modularidad soporta actualizaciones/despliegues independientes que ayudan a evitar los puntos de fallo, lo que previene las interrupciones a gran escala. Existen dos visiones del escalado [221], en base a los recursos que se van a utilizar y en base a los servicios que se van a proporcionar. En el primero de los casos podemos entender que existen los siguientes tipos, mostrados en la Figura 2.11:

- **Horizontal Scaling**. Se logra modificando el número de servidores o microservicios existentes. Se dice que se produce un *Scale In* (-) cuando se reduce el número de nodos o un *Scale Out* (+) cuando se aumenta su número. Su principal ventaja es que este tipo de escalado es casi infinito, pudiendo añadir un número muy elevado de elementos adicionales. Su principal desventaja es la necesidad de adaptación de las aplicaciones. Por ello, deben estar construidas para soportar esta característica.

- **Vertical Scaling**. Se entiende que se actúa sobre los recursos hardware o virtualizados que tiene el nodo de procesamiento. Así se produce un *Scale Down* (-) cuando se lleva a cabo una reducción de recursos y un *Scale Up* (+) cuando se aumenta cualquiera de los recursos disponibles para su uso. La principal ventaja de este tipo de escalado es la facilidad de su implementación y configuración y su principal desventaja es su limitación por el hardware físico subyacente.

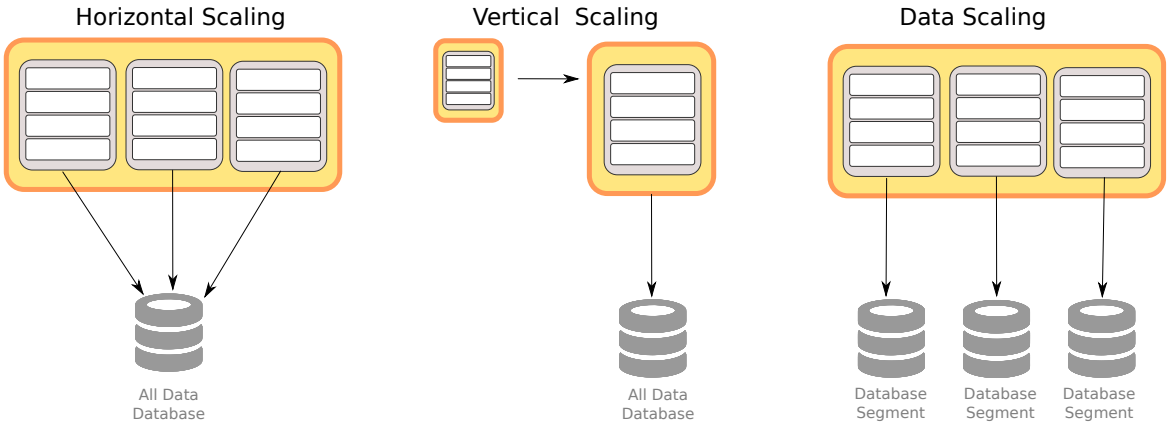


Fig. 2.11 Tipos de escalado.

Como un caso independiente podemos ver el *Data Scaling*. En tareas relativas a procesos de *machine learning* o de *data mining*, el *data scaling* se refiere al preprocesado de los datos y se utiliza como sinónimo de normalización [24]. En el mundo del Big Data, también se le conoce como *data segregation*, y se entiende como la separación de los datos para su envío a diferentes nodos o grupos de nodos. De esta forma los datos se dividen por múltiples criterios y son procesados de forma más eficiente. Este proceso no esta exento de problemas, como pueden ser las dependencias entre los propios datos. En ocasiones es una tarea que puede estar asociada o ser complementaria al *horizontal scaling*. En el ámbito de las bases de datos esta técnica se conoce como *sharding* y se utiliza como mecanismo de escalado de bases de datos relacionales.

Otra forma de ver el escalado es la propuesta de Abbot et al. [1], mostrada en la Figura 2.12, y que aporta un método de escalabilidad basado en tres ejes denominado *Scale Cube*. Esta propuesta nos puede ayudar a identificar las oportunidades para el aumento de la escalabilidad de un sistema:

- **X-axis Scaling.** Ejecuta múltiples copias de la aplicación en un conjuntos de datos compartidos como se muestra en la Figura 2.13a. Todos los servicios son capaces de ejecutar los mismos programas. Las cargas de trabajo son manejadas por un balanceador de carga. Es la opción más sencilla pero requiere más memoria caché y no escala bien en aplicaciones complejas.

- **Y-axis Scaling.** Se divide la aplicación en varios servicios, cada uno de los cuales es responsable de una funcionalidad específica. En la Figura 2.13b un balanceador encamina las peticiones a la tarea adecuada para dar servicio exclusivo sobre ese área.

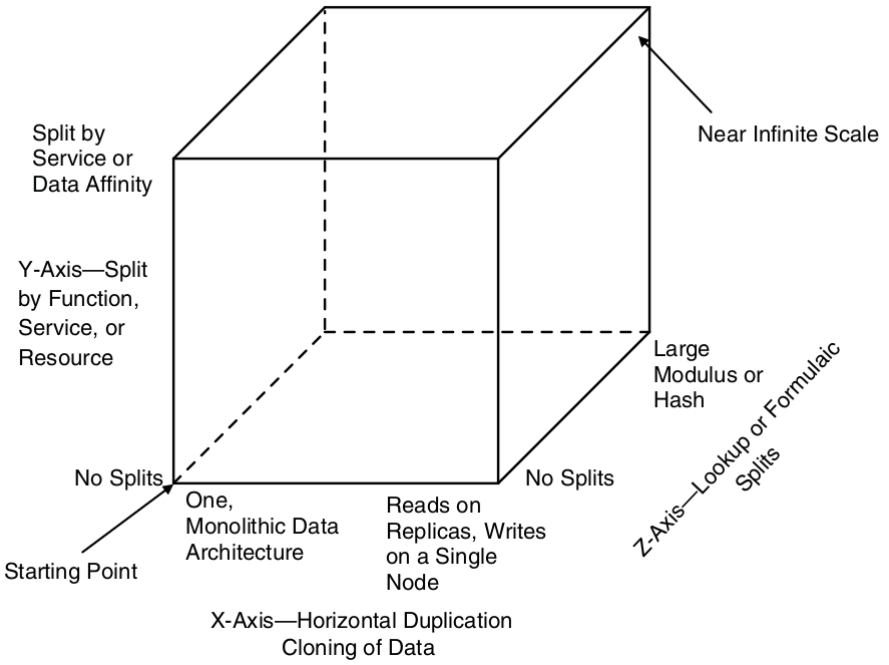
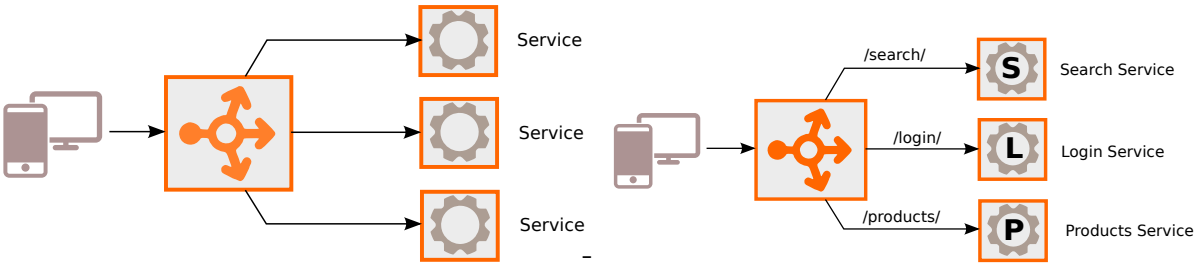


Fig. 2.12 Dimensionamiento en tres ejes. *Scale Cube* [1].



(a) Escalado X axis. (b) Escalado Y axis.

Fig. 2.13 Tipos escalado X-axis e Y-axis.

- **Z-axis Scaling.** Es similar al escalado Y-axis pero en cada caso solo se maneja un subconjunto de datos utilizando una clave primaria con objetivo de segmentar las rutas. En la Figura 2.14, un balanceador se encarga de enviar el contenido a la partición apropiada. Esta aproximación es más eficiente en función de la memoria y más escalable en escenarios transaccionales.

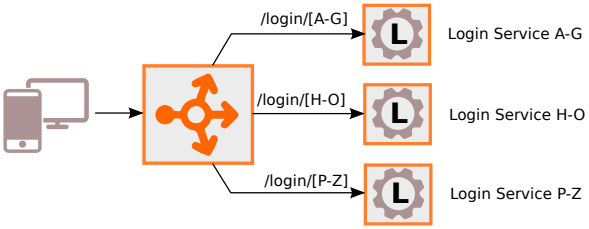


Fig. 2.14 Escalado Z axis.

Sobre estos tres modelos de escalado básicos se pueden diseñar múltiples combinaciones para hacer frente a peticiones más complejas. Entre las posibles combinaciones, una de las más básicas es la que mezcla X-axis e Y-axis y que se muestra en la Figura 2.15.

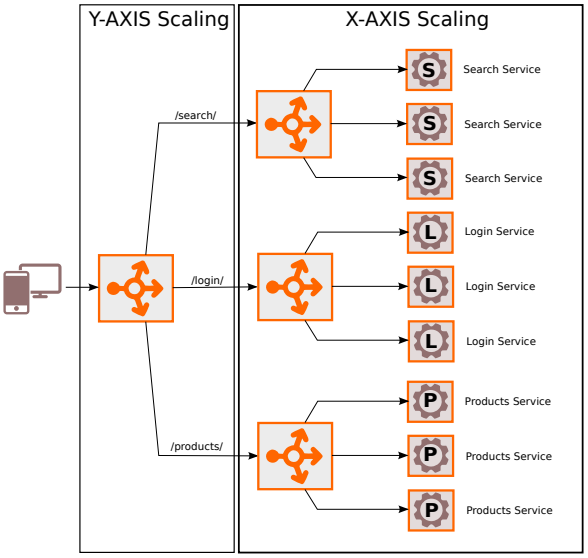


Fig. 2.15 Combinación X-axis e Y-axis.

2.4 Otras arquitecturas

No quedaría completa la panorámica de las arquitecturas de procesamiento de Big Data si no se realizara una mención de algunas de las arquitecturas con menor implantación

pero que son aún destacables [221] ya que conforman el estado actual de estos tipos de tratamientos [206] [130]:

- **Arquitectura Zeta** (Z). Propuesta en el trabajo de Scott [215] en 2015. Esta propuesta de arquitectura tiene como centro el dato [168]. Solventa algunos problemas que las arquitecturas de Big Data tradicionales no resuelven, especialmente en relación con el uso eficiente de los datos. Es una arquitectura holística con 7 componentes [143] [168]:

  1. Sistemas de archivos distribuido. Es una ubicación común de datos para todos los procesos. Con características de escalabilidad y confiabilidad avanzadas.
  2. Almacenamiento de datos en tiempo real. Basada en tecnologías de tiempo real es responsable de entregar respuesta a procesamientos de datos en tiempo real.
  3. Capa de aplicaciones empresariales.
  4. Arquitectura de soluciones. Combina aplicaciones para entregar una aplicación completa. En este área podemos encontrar algoritmos, componentes software y flujos de trabajo empresariales.
  5. Modelo de cálculo o ejecución. Ofrece motores y modelos de ejecución para satisfacer las necesidades de procesamiento.
  6. Administración dinámica y global de recursos, donde se incluye la asignación dinámica de recursos como Apache Mesos o Apache Hadoop YARN.
  7. Sistema de despliegue de contenedores. Garantiza un método de implementación único.

- **Arquitectura Delta** (Δ). En este ejemplo de arquitectura se asume que los registros de entrada son procesados como registros incrementales (delta). Conceptualmente es similar a Lambda centrada en el *hot path*. La principal diferencia es que no considera los *data lake* o almacenes de datos como inmutables. A diferencia de otras arquitecturas las transformaciones *batch* pueden actualizar la estructuras de estos repositorios. Esto hace que el *cold path* sea mas sencillo de procesar [103] [153].

Podemos ver que existen otras posibles opciones para definir la estructura de los procesos de Big Data. En algunos casos encajan con los estudios realizados, como puede ser la arquitectura Zeta que incluye, entre otras, todas las partes de este proyecto de investigación, pero siendo en estos momentos solo una propuesta inicial.

2.5 Bases tecnológicas de los algoritmos utilizados

En el transcurso de los trabajos se utilizan, y durante esta tesis se hace mención, a varios algoritmos y técnicas que se adaptan a las necesidades de los desarrollos realizados. En esta sección se mencionan brevemente aquellos más importantes con el fin de tener una breve referencia de los mismos.

2.5.1 Transformación de Burrows-Wheeler

La transformada de Burrows-Wheeler (BWT) [19] es un algoritmo de ordenación de cadenas reversible. Los datos de salida son los mismos símbolos que los existentes en la entrada pero ordenados de una forma diferente. Este cambio en la ordenación de las cadenas es lo que permite una compresión eficiente pero al mismo tiempo permite localizar todas las coincidencias de un patrón agrupadas en la matriz de BWT.

Una de las propiedades que se puede destacar, y por el que es utilizado en el área de la bioinformática, es su capacidad para localizar cadenas de ADN procedentes de secuenciadores de alto rendimiento. Estos sistemas, aparecidos en la década del 2000, generan millones de lecturas de una longitud de entre 30 a 500 pares de bases pertenecientes al ADN que se analiza. La tarea en la que funciona eficientemente la BWT es la de alinear estas lecturas a un genoma de referencia que puede tener varios miles de millones de pares.

En base a esta transformación se han desarrollado otros algoritmos de alineamiento adaptados específicamente a las necesidades existentes como pueden ser Bowtie [151].

2.5.2 Bloom-Filter

Es una estructura de datos probabilística basada en funciones de *hashing*. Diseñada originalmente por Burton Howard Bloom en [16]. Se utiliza para verificar que un elemento se encuentra en un grupo. Las características probabilísticas del filtro hace que tengamos la posibilidad de obtener negativos o bien falsos positivos pero nunca obtendremos falsos negativos. Debido a que se utiliza una estructura de datos reducida muchas claves ocuparan el mismo espacio, siendo este el motivo de los falsos positivos. A cambio, tenemos un sistema extremadamente eficiente en cuanto a espacio y tiempo necesario para la consulta. Es adecuado ya que se mantiene constante el rendimiento ante un aumento del tamaño de los datos.

2.5.3 Diarización

Las diarización no es un algoritmo desarrollado, sino que es una técnica que permite realizar una tarea. En nuestro caso el objetivo de la diarización es la de detectar e identificar los cambios de locutor existentes en una grabación sonora con el fin de lograr la indexación automática de contenidos del audio. De esta forma, después del proceso, debemos obtener segmentos de voz identificados con los diferentes locutores existentes en un fragmento sonoro.

La utilización de sistemas de reconocimiento de locutores en estos momentos pertenece a la rama de la inteligencia artificial. Las voces de las personas se pueden distinguir mayoritariamente por sus características fisiológicas pero también por sus hábitos lingüísticos. De esta forma los algoritmos se puede separar en dos fases: identificación de las características de los locutores, asociada generalmente con la sección de entrenamiento, y las sección de clasificación donde se identifican las porciones en el fragmento.

2.5.4 Dynamic Time Warping

El algoritmo DTW (Dynamic Time Warping) es conocido en varias áreas desde su creación en los años 60 [207]. Ha sido utilizado con multitud de fines [217] (reconocimiento del habla, de la escritura manual y reconocimiento de firmas, lenguaje de signos, reconocimiento de gestos, minería de datos, visión y animación por computadora, vigilancias procesamiento de música y señales, etc.) aunque en la actualidad se han encontrado algoritmos más eficientes para algunos ellos.

Es conocido por la comunidad ya que es extremadamente eficiente y se puede obtener una medida para la similitud de series temporales minimizando el efecto que tienen los desplazamientos temporales y la distorsión en el tiempo de las dos series. Esto permite comparar series que poseen diferentes fases.

En esta tesis se maneja con el objeto de comparar series temporales para confrontar la respuesta de autoescalado de los sistemas diseñados contra una respuesta ideal previamente calculada.

2.5.5 Algoritmos bio-inspirados

Los algoritmos bio-inspirados los podemos agrupar en tres grandes bloques: algoritmos evolutivos, inteligencia de enjambre o basados en la ecología [203]:

- Los algoritmos evolutivos están basados en las leyes de la selección natural: la selección, las mutaciones y la reproducción. Todos ellos se basan en aplicar criterios de supervivencia de los mejores en un determinado ambiente (problema a solucionar) y que dan lugar a la siguiente generación.
- Los algoritmos de inteligencia de enjambre se basan en el estudio del comportamiento de los insectos sociales como pueden ser hormigas [208], abejas o incluso pájaros. La base de todos los algoritmos es la interacción de los agentes, con un comportamiento simple, en base a unos principios básicos: principio de proximidad, principio de calidad del entorno, principio de respuesta diversa, principio de estabilidad y principio de adaptabilidad. De esta forma, aparecen algoritmos bien conocidos como la optimización del enjambre de partículas, la optimización de la colonia de hormigas o el algoritmo de la colonia artificial de abejas.
- Los algoritmos dentro de la categoría de ecológicos se centran en la interacción de los seres vivos con el ecosistema en el que habitan. Las relaciones con otros individuos, con el medio o con otras especies es la norma para realizar evoluciones.

Con estos dos últimos bloques en mente, se realizó un emparejamiento entre los valores del ecosistema y parámetros del microservicio que dio lugar a un sistema que evoluciona en función de los parámetros del "ecosistema informático" dando respuesta a las necesidades de autoescalado en base a la evolución de la población.

Capítulo 3

Studying the Improving of Data Locality on Distributed Grid Applications in Bioinformatic

Título:	*Studying the improving of data locality on Distributed Grid Applications in bioinformatics*
Autores:	*José Herrera e Ignacio Blanquer*
Publicación:	*7th IBERIAN Grid Infrastructure Conference Proceedings. 2013.*
Licencia:	*Creative Commons Attribution-ShareAlike 3.0 (CC BY-SA 3.0)*
Referencia:	*[105] - https://riunet.upv.es/handle/10251/31992*

El primer artículo [105] se centra en una optimización del uso del sistema Grid de EGI (ver en sección 2.1) mediante la utilización de técnicas de descarga en paralelo y/o caches locales de datos con el fin de acelerar uno de los procesos de alineamiento de cromosomas más utilizados. Con la transformada de Burrows-Wheeler (BWT) [19] se logra aumentar la velocidad del alineamiento de las porciones del genoma procedentes de los secuenciadores de última generación. En este artículo se optimiza este proceso de búsqueda para obtener mejores resultados en la infraestructura Grid mencionada.

El proceso inicial, que intenta mejorar, se basa en la descarga de los datos genómicos de la *Drosophila Melanogaster* para después modificar la estructura de datos mediante la transformación BW. Esta transformación, que es costosa en el tiempo, se repite en cada uno de los nodos donde se precisa alinear un fragmento. Posteriormente a la

transformación, la búsqueda de los fragmentos de la secuencia se realiza de una forma óptima y eficiente.

Se propone mejorar el sistema de entrada/salida del Grid empleado por los experimentos mediante la extensión de una de las librerías más utilizadas: GFAL2 [196]. Ésta es una librería C que proporciona una capa de abstracción superior a la complejidad de los sistemas de almacenamiento del Grid. De esa forma se prevé incrementar el rendimiento en los procesos de lectura de ficheros remotos y mejorar el uso de múltiples fuentes de datos, aplicando estrategias para la reducción del ancho de banda y utilizado todo ello de forma transparente al usuario/proceso final.

El algoritmo de alineamiento utilizando (BWT) es un proceso secuencial donde se obtiene la transformada y después es utilizada para buscar la secuencia genómica asignada en el procesamiento en múltiples nodos. Es por ello que tener esta transformación ya realizada y mantenerla en cache permite la mejora del sistema Grid.

Con la idea de valorar las mejoras que se producen al emplear diferentes enfoques se diseñan seis estrategias de prueba que son las siguientes:

- Estrategia 1. Basada en la utilización de *Storage Element* (SE) más cercanos para mejorar los tiempos de inicio efectivo del proceso de alineamiento.
- Estrategia 2. Se centra en la compresión de los ficheros necesarios previamente a su almacenamiento. Esto implica también un proceso de descompresión previo a su uso que retrasa el inicio efectivo de los procesos.
- Estrategia 3. Donde se emplea la descarga *multi-stream* lcg-cp para un mismo cromosoma.
- Estrategia 4. Se paraleliza la descarga de los ficheros de un mismo origen con el fin de obtener mejores tiempos de carga sin utilizar lcg-cp como en la estrategia anterior.
- Estrategia 5. Se emplean varias fuentes de datos y se elige el origen que ha sido definido históricamente como el más rápido.
- Estrategia 6. Ajustada para no descargar el fichero del cromosoma en su totalidad al principio del proceso sino que se emplea el acceso remoto para el tratamiento en combinación con la aplicación de técnicas de cache locales. La descarga se produce en el momento que es necesario el dato.

Se pueden apreciar mejoras en los tiempos de ejecución dependiendo del caso aplicado. En la estrategia 1, se constata que si se utiliza SEs más cercanos se mejora la

ejecución. De la misma forma se puede constatar una mejora del proceso que combina descarga+descompresión (estrategia 2) en comparación con la descarga sin compresión. Es en la estrategia 6 donde se obtienen mejores resultados. Analizando el patrón de acceso del proceso de alineamiento se observa que solo un número muy reducido de bloques del fichero remoto transformado es accedido, lo que permite, que con una estrategia que no obligue a descargar los ficheros en su totalidad, se reduzcan los tiempos de procesamiento. Si a esta estrategia le añadimos un acceso optimizado con una cache que respeta el principio de localidad de referencia, podemos obtener mejoras apreciables.

Abstract

Data locality severely affects performance on Grid distributed applications. Many applications require accessing remotely stored large blocks of data over thousands of kilometers. The replication of data along the Grid infrastructure reduces the risks of bottlenecks and increases availability. However, proper selection of the rightmost source of data is not automatically provided by the infrastructure. Moreover, a number of applications running on the Grid require the retrieval of small sections of reference files which ought to be copied locally. EGI middleware provides APIs which enable the opening of files and fetching portions of datesets, although performance is affected when accessing multiple random blocks. This paper proposes an extension that provides a better selection of the rightmost storage resources, according to historical records and network criteria, combined with the ability of caching blocks of data retrieved through GFAL calls. In order to validate the model, an experiment has been carried out using a nucleotide alignment code based on the Burrows-wheeler transform, widely used in modern Next Generation Sequencing data.

3.1 Introduction

In a highly distributed environment such as Grids, data access constitutes a major element affecting the performance of Grid applications. Modern Grid infrastructures rely on the replication of data to reduce bottlenecks and increase the chances of approaching computing power to data. Data catalogues can keep indexed several copies of the same data, providing in a round-robin or random fashion the actual location of a logical file. However, this approach faces two relevant restrictions: Firstly, data files should be either retrieved completely or manually split into smaller chunks in case

that only partial access is needed; secondly catalogues do not provide the rightmost location for each job depending on the bandwidth.

The first issue is partially overcome by software libraries such as GFAL [69], which exposes a POSIX-like interface to applications. Thereafter, applications can open files, seek pointers and retrieve only the bytes required, overriding the need of downloading the whole file. This may also facilitate the migration of applications which can be easily modified by pointing to external locations. In addition, GFAL supports the access to local files with the same API. Nonetheless, GFAL access to remote files is costly and may not be efficient. In the case of high data-locality access patterns, GFAL may be combined with caching strategies which could considerably improve performance. This is one of the key points tackled by this paper. Other approaches consider using automatic caching systems [197] for the whole access to remote files. Despite that this will be less intrusive, we want to concentrate in an approach in which the programmer may have deeper control.

The second problem might be better dealt with at two levels. At first level, the GLUE schema [18] provides a way to define the closest Storage Element (SE) to any given Computing Element (CE) which manages the batch queues that will effectively run the jobs. If a replica is available at the closest SE, then the retrieving APIs could be amended to get the most suitable replica. However, if no replica is available at the closest SE, then the choice is not trivial. The best approach would be to keep periodic records of the latency and bandwidth performance among the different pairs CE (actually, the working nodes (WN) of the CE) and the SEs. This can be kept distributed and continuously updated to keep track of the performance.

This article presents an experiment to measure these two approaches and presents early conclusions on the results. The article presents also a proposal of software architecture to implement data caching and network performance recording for the matchmaking of the most suitable replica in the storage. The article is structured as follows. This section introduces the issues and the approaches. Section 3.2 describes a proposed architecture to improve data locality. Section 3.3 describes the use case, and section 3.4 describes the experimental results. Section 3.5 lays down the conclusions.

3.2 Architecture overview

To improve the current Grid I/O system, we propose a new architecture gfal-CE (Grid File Access Library Cache Extension) with new enhancements based on temporal and

spatial locality. The present article describes the proposal of the architecture and the results of a set of experiments which will justify the suitability of the approach.

Figure 3.1 shows the proposal to reduce remote file access impact. The design of architecture is guided by the operational and performance requirements shown below:

- Increase performance in remote file read operations.
- Reduce, on cache miss, the waiting time.
- Uses multi-source files transfer to reduce transfer time.
- Applies strategies to decrease bandwidth use.
- Ensures and supports common GFAL access API.
- Transparency hides algorithms complexity for the end user.

We define temporal locality as the probability that at one point in time a memory location is referenced by the process; thereafter it is likely that such location will be referenced again sometime in the near future. Similarly, spatial locality can be defined as the probability that when a memory location is referenced at a particular time, nearby memory locations will be referenced in the near future. The use case proposed in the article, as it can be deducted from the experiments, is a good example of both temporal and spatial locality.

Nowadays in most computer systems scenario, there are numerous examples of cache systems. From L1 and L2 system cache in multicore processors to proxy cache servers that store web pages (in order) to be delivered to web browsers in a fast and efficient fashion. In Grid systems, denote Dcache, there is a cache system to speed up the retrieval of slow-performing data storages. Some free proposals have been found such as [191]. However, currently EGI clients' side lacks of a cache system to improve data transfer features.

We suggest that a process can use a new GFAL-API when it may benefit from caching data. This new layer would spot if the file block is locally saved. In such a case, data is delivered to process. When a file block is not in a local position, the new layer looks elsewhere for data block in close SEs. If this second retrieval attempt fails then it downloads the file block using GFAL classic API. In such a way, our proposal is a two level cache system to be used prior to the remote access. The first level would be associated with local process and local data, while the second level would be associated to EGI infrastructure provider facilities.

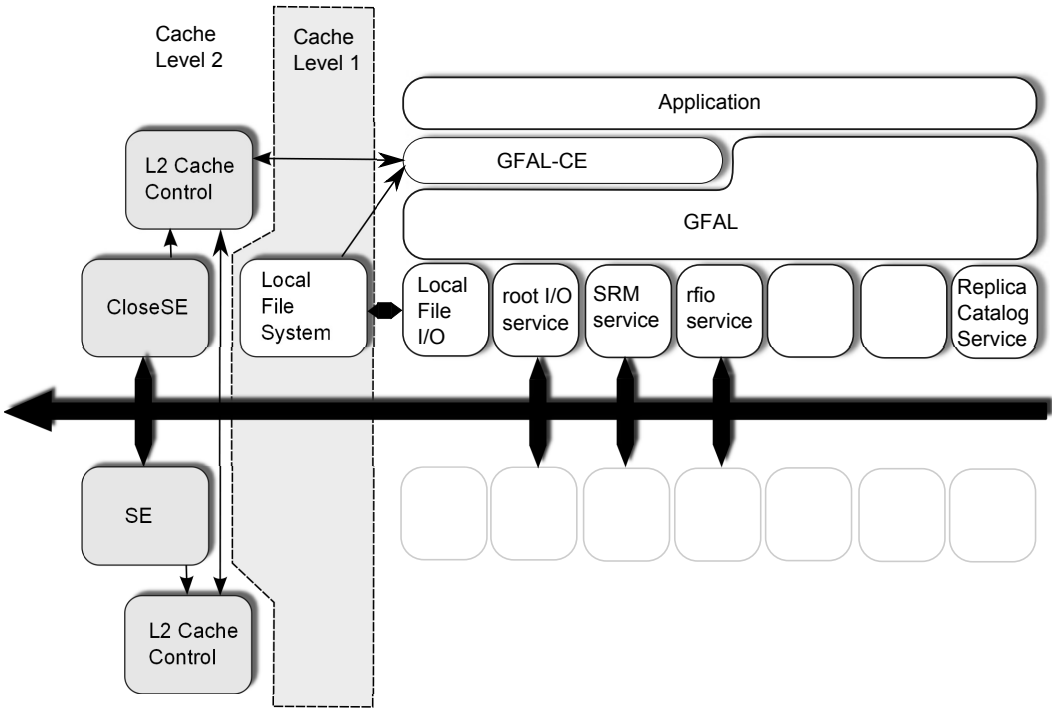


Fig. 3.1 gfal-CE Architecture preview.

A file must be divided in chunks that are transferred using the GFAL API to services provider and local node cache. This way, it is possible to share files between different nodes. This would overcome the need of physically splitting files, which ultimately has an impact on the code execution. Having said that, further effort is needed to design in the new API a conversion system to map out real access to block access.

The first level cache saves file blocks in local CE disk infrastructure. Blocks are fetched and downloaded for each data miss operation. No data is shared between processes that do not share the same file system. Only threads running in the same processor must use the same first level cache data. Of course, saving data to the file system increases the cache size.

When an application data request cannot be met by the local cache the gfal-CE layer calls a second-level control cache server. Control cache is not the supplier of data. Its first function is to determine the position of the file blocks that are stored in the SE and and also achieve bandwidth reduction strategies. If data is not in the second-level cache then it must be downloaded using different techniques such as deduplication [141], delta encoding [175], compression, etc. or using GFAL API and/or using file replicas or stripping. Once the block download is over, the request is served. Next request to the same block will be served quickly. Second-level cache is shared between

all CEs running in a NGI infrastructure provider. When a file is used by a CE, next time that a different CE uses the same file, it will be served by the cache system. A second function of the control cache server is to download data in anticipation of data request for jobs.

Gfal-CE layer is located above GFAL. The most suitable solution would be to extend GFAL API. It is necessary to maintain previous GFAL API to migrate some code and develop new code as soon as possible, so that by extending the API interface with new primitives, efficiency is increased. Only processes that use gfal-CE will use the cache services, whereas other processes using original GFAL library do not benefit from such improvements. Therefore, two proposals are able to coexist in the same infrastructure. Jobs may use the new library not requiring changes in the infrastructure level, by providing a second lightweight layer.

Reading operations in files are typically greater in number than writes, which do not happen on the reference data of the use case proposed. Therefore, our first proposal overlooks writing operations in the cache system. A feature in gfal-CE and other cache system is that cache size is limitless. Hard Disk storage in either SE or CE are far greater than in other cache proposals with limited space.

A cache system could hide the complexity and access data needs and prevent needing to download a whole file previously, either through scripts or the job description language (JDL) sandboxes. Extending GFAL (using gfal-CE) avoids this error source and, thus, can increase the fault tolerance of the entire system.

3.3 Use case: Burrows-Wheeler alignment

The so-called Next Generation Sequencing (NGS) constitutes a new type of sequencing devices which are providing an unprecedented large of raw data but in a more fragmented fashion. These features (billions of fragments below 200 basis) created the need for new methods for alignment and processing. Among them, the Burrows-Wheeler transformation [19] and the FM-Index [61] have been intensively used for the implementation of tools such as Bowtie [151], BFAST [111][112], SOAP [159], HPG-Aligner [56] and BWA [158].

BWT-based alignment is an embarrassingly parallel process that takes a variant number of input fragments and compares them with a reference (normally an existing consensus genome). It uses three pre-computed data structures from the reference genome and the computational cost depends on the length of the input sequences to search for and the number of errors (freedom degrees) allowed. Details on the

algorithm can be found at [158]. The reference data structures are huge (in the order of more than 40 bytes per nucleotide base), so different techniques are used to reach a compromise solution among memory restrictions, performance and accuracy. In a restriction-free model, each run will access to a limited number of elements in the reference data, depending on the size of the input data and the number of errors. However, those accesses cannot be predicted a-priori, thus requiring the whole reference data. The subsequent accesses of the algorithm for the different basis of each sequence are normally located in different areas of the reference data.

However, when processing input sequences that have common prefixes, the same elements of the reference data will be required. If data is somehow sorted, the changes of reusing the same data increases. Reordering the whole 200-length sequences may be prohibitive, but sorting only by the first basis will require a reasonable computing cost. This would suffice for increasing the hits on accessing elements of the reference data.

An out-of-core version of the BWT algorithm for the alignment of an input sequence with respect to a reference has been implemented based on [56] and [158]. This algorithm stores directly the three main pre-computed data structures (the BWT transformation, the number of occurrences in each basis at each position and the reference genome) in three different files. In the case of the Human Genome [98], the total size required will be above 100 GB. The experiments have been carried out with the genome of Drosophila Melanogaster [26], which requires substantially less memory. When accessing a single element in these data, the algorithm checks if the data is available and if not it fetches a block into the local disk. The elements can be randomly accessed since the position is known.

3.4 Experimental testing

The experiments aim at verifying the hypothesis posed in the introduction (caching chunks will have a positive effect on performance and SE selection will have an important impact on performance). These experiments will also identify key parameters that will be used to tune in the architecture. The experiments will cover the execution of an out-of-core version of the algorithm with different configurations, comparing to the execution of the same case with an on-memory application. For the out-of-core case, several strategies will be tested. We start downloading the whole file, with and without multistreaming. Multistreaming will be consider both at the level of the APIs and manually. Additionally, we measure the importance of choosing the nearest (in terms of bandwidth) site. Finally, we implement a last version using directly the

GFAL software library to analyse the potential impact on the locality of reference. The different experiments are described next.

3.4.1 A first and key experiment

Using the BWT algorithm described above, we first developed a two-part implementation to reduce memory requirements. Based on a recursive search code developed in [228], we coded a new two step program. For comparison purposes, we need other implementation running under the same conditions.

Now we present the general conditions for the first experiment. It consists on two main data files: the sequence file to search and the reference data file that contains the known data. The search file contains only one line 200 nucleotides wide (Sequence A). The second one used the sequence of the chromosome one from HG19 Homo sapiens in FASTA format from [26]. This implies a search space of 254,235,634 nucleotides. We allow up to 2 mismatches in the searching process.

The execution of the original code (BW_search) has a runtime of 864s using a Intel Core i5 760 with 8 GB memory and swap file employment around 16 GB. The total memory for execution was 24 GB. Both main memory and swap file have high occupation level. Due to high memory usage we are trying to compare execution time only in the local processor.

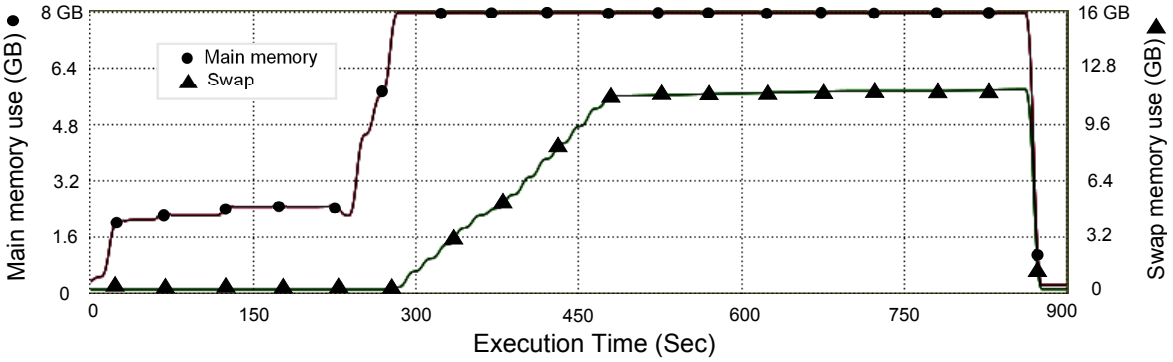


Fig. 3.2 Memory use for BW_search.

With the purpose of reducing the total execution time in BW_search and decreasing the total memory used in search process, we divide the code in two steps: BW_build and BW_quest. BW_build writes files in hard disk with the BW transformation and all the data search. BW_quest uses the files created in a previous step for the search algorithm.

BW_build, the first step, generated four files with data structures that before were in memory. All big structures data was stored in separated files that were uploaded to a SE in the grid infrastructure if it was necessary. Total execution time in test environment was 353s. BW_build has independency from one of the parameter, the search file.

BW_quest uses the files created in BW_build to search chain location. BW_quest will not load files for the supported structures in main memory. Every access has been converted to file accesses in order to reduce the total use of RAM memory. BW_quest uses only one parameter search file. Almost all operations were reading.

In BW_quest, obviously, the use level of memory is reduced. Total time for the execution of BW_quest is 472s . The execution time is reduced since BW_build is executed one time and intermediate results remain unchanged as reference data remains unaltered. Therefore, in order to evaluate and measure the execution time in the new solution (BW_quest) we divide the process in three parts:

QuestA - Download source files time from SE to WN and have the data ready for running BW_quest.

QuestB - Process BW_quest time.

QuestC - Upload result file time. Save result data in SE.

(iii) Mesuring time for QuestC. Usually, the size of the resulting file will be directly depending on the size of the input data. Thus, in our case the result file will be of a reduced size. If we have the same parameter on both sides of the equation then we can disregard it. BW_search and BW_quest have the same result report. We always consider this time zero.

(ii) Mesuring time for QuestB. QuestB time is the execution of the algorithm in a single processor without file transfer. This step measures algorithm speed compared to the original algorithm. Using the same testing conditions, BW_search execution time is 472s . Time improvement becomes 45% better. Total executing time with this enhancement is around 55% of the previous total executing time. If we pay attention, the profit occurs if the creation of the intermediate data file only occurs once, and it keeps these data in a non-volatile structure for reuse. A large amount of time was used to create search and sort Burrows-Wheeler Transform matrix.

(i) Mesuring QuestA time. This is the time required to move input data files from SE to WN. In our proposal, we downloaded all files required to run the process. Intermediate files always have a total size around 4 GB. In order to measure QuestA time, it is necessary to define a proper strategy. Different strategies are being considered in different experiments and results are shown in the following sections and in the table 3.1.

Strategy 1. Sequential download of files. We tested two source locations. The nearest CE, sometimes referred to as the CloseSE, and other node with a good response time. As expected, the result with the files stored in the CloseSE are better. The result is a download time of 259 seconds ending up with a total execution time of 731 seconds (259+472) for group QuestA+QuestB time.

Strategy 2. Get compressed files and uncompress them before running. A second approach to the solution is to compress the files before downloading them. In this case, QuestA time includes both downloading and uncompressing time. Gzip unix command is used for uncompress step. Results are similar to the ones in the previous table but with lower time. The total execution time: 364s (115 download + 250 uncompress) added to execution time is 836 seconds (364+472).

Strategy 3. Get files using multi source access from two or more source locations. We used the multi-stream lcg-cp option but we could not make any download from distant nodes. We note that multi-stream lcg-cp download requires certain port range open in the firewall on the destination host. The performance is slightly better in this case. Total execution time was 726s (254+472).

Strategy 4. Parallelize file download. Reference data can be divided in two or more parts, which can be manually downloaded in parallel to reduce total time. For testing purposes, in this case we downloaded every file at the same time using a parallel job running lcg-cp process. Total time 685s (213+472).

Strategy 5. Select best SE. We have replicas in several geographically distributed SEs. Keeping historical records and selecting the fastest node for downloading is a right strategy. This approach is described in the second experience where we try to find a way to establish the best download node.

Strategy 6. Access the file remotely using GFAL, avoiding downloading the whole file. The best way to reduce the downloading time is to reduce the total number of downloaded bytes at QuestA and try to access remotely to only the required information.

Strategy	Streams	from SE ngiesse.i3m.upv.es	from SE se01.dur.scotgrid.ac.uk
1	1	259s	441s
2	1	364s	474s
3	2	255s	—
3	5	254s	—
3	10	261s	—
4	1	213s	339s

Tabla 3.1 Strategies estimated times downloading to ngiesui.i3m.upv.es.

3.4.2 A second experience. Describing the 5th strategy

Files used as intermediate storage system increase the data time access and made file access one of the purposes of the improvement process. We introduce a second idea, file replication. When there are many replicas in grid, a copy is most likely to be available. This idea is normally used in grid jobs to reduce bottlenecks. The files were distributed along SEs nodes and the job running in a WN locates and transfers files to a local storage space.

To evaluate access time and availability of a file in the biomed VO SEs, we designed a test that downloads replicated files from SE and saves access time and total errors.

We measured access time as the SE response time and its bandwidth. The experiment is extended reducing the size of the file and, therefore, decreasing the requirements on bandwidth. For testing purposes, files measure 4 bytes. This way, the infrastructure is evaluated in terms of latency (speed in communication lines). The experiment consists of performing several file downloads using the lcg-cp command. This command tries to download the file from different servers and it applies a recover algorithm to download them from other server if some error occurs.

The file was replicated in six SEs selected by their distance to running environment (grid130.sinp.msu.ru, ngiesse.i3m.upv.es, se001.ipp.acad.bg, se01.marie.hellasgrid.gr, svr018.gla.scotgrid.ac.uk and tlapiacalli.nucleares.unam.mx).

Default time-out values for lcg-cp command were: for BDII access 60s, for connection 60s, for total time 3,600s (1 hour) and send or receive ratio of 60 blocks. The number of errors is reduced with these values. Lower time-outs led to higher error rates.

After the test run, it is observed that error rates are randomly appearing on different machines. At 05/01/2013 to 05/07/2013 week transfers time between SE an UI has an error ratio of 15-18%. However, during the following week error ratios fell down to

zero. In some cases, time access errors have been caused not by the SE slowness but by the BDII. If we are more demanding in BDII performance, then total errors will raise. We note that response time is not a function of the physical proximity and is not a constant during the experiment life-span.

The conclusion of this simple experiment is that if we would like to consider the response speed of the node containing a file replica, the values must be dynamic and they must be updated frequently. Also, it indicates that the measurement of access time in the transfer of files is not constant.

3.4.3 A third experiment. Explaining the 6th strategy

In this third experiment we try to direct access to remote data. Before that, in order to learn more about BW_quest I/O behaviour, we did some tests. Using iostat linux command we measured I/O access for BW_quest. Analyzing data obtained, it highlights that read access are grouped around a short timeframe.

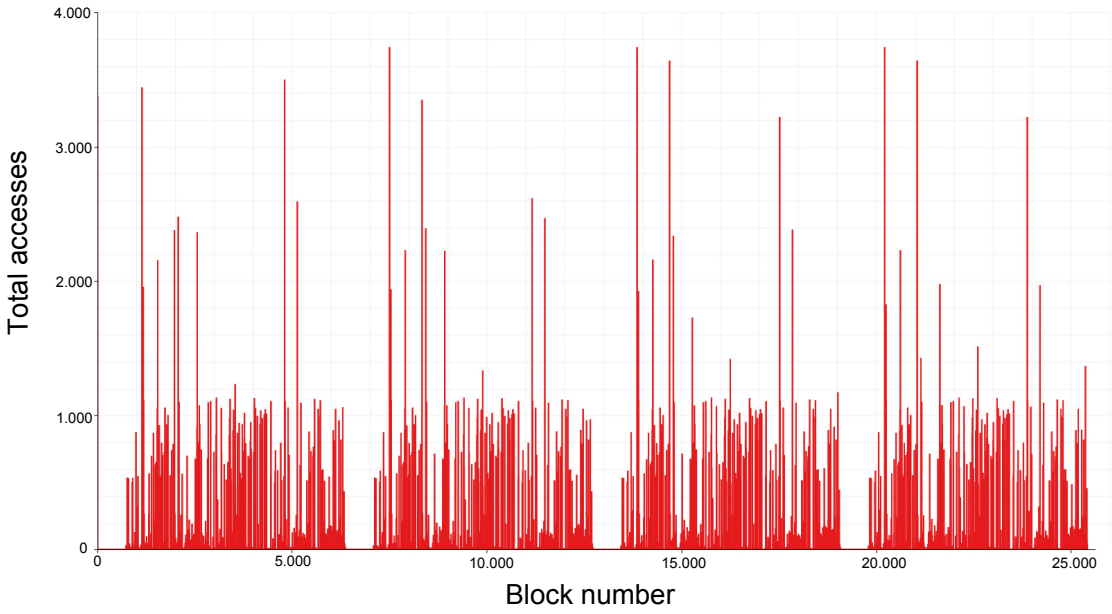


Fig. 3.3 Total accesses by block (Sequence A).

In order to clarify what happens in the read I/O period, we have altered the original code for previous experiment BW_quest. We need to know read positions executing. Therefore, we altered the code to report I/O from local access saving it in a log file. We selected only accesses to the largest file (file_o.txt) and, in order to show data after

capture, we created groups of total access. The group size is 4,000 integers. This means that if job reads an integer from 1 to 4,000 positions then this access is represented in the first column and also the remaining ones.

We created 254,235 blocks. In figure 3.3, we represent block access number. A detailed analysis shows us that the execution uses a small number of blocks. A total of 24,541 blocks are accessed from a total of 254,235. Only the 9,6% of the blocks are used. If file_o.txt, a large intermediate file, size is 3,7 GB then only 364 MB are necessary for the algorithm execution. We also note a replication pattern that exists in the blocks access. There are four access big groups that have a similar pattern.

Finally, we try to access remotely. A 4 GB transfer has approximately 254s cost tested in EGI infrastructure that it is added to compute time (QuestB time). At this point, we would suggest not to download all files prior to calculation, but access the data remotely. In order to exploit this feature, we altered BW_quest to access directly to data file using GFAL API. When we use gfal_seek and gfal_read commands for data access, it is not necessary to download the file. This reduces files download time (QuestA time) but it increases drastically QuestB time. Capture time, for one search, extends up to more that 1,500s.

The table 3.2 summarize the behaviour for each strategy.

	QuestA	QuestB	Total
BW_search	–	–	864s
Strategy 1 - Sequential	259s	472s	731s
Strategy 2 - Compressed	364s	472s	836s
Strategy 3 - Multistream	254s	472s	726s
Strategy 4 - Parallel	339s	472s	685s
Strategy 5 - Best SE	200s	472s	672s
Strategy 6 - Direct Access	–	1,500s	1,500s

Tabla 3.2 Estimated access time for one sequence alignment.

The best way to make out access pattern for sequences alignments is to compare two similar sequences. We choose two similar sequences, A and B, it shares the 32 first bases, from position 1 to 32 was the same, position 33 and later was different in some places. In table 3.3, we compare some data from the two accesses. If we run sequentially, first A and then B, we note that total blocks read by the two processes were 599 and only accessed by one process are 23,942 blocks. All blocks used by B sequence were accessed previously by sequence A. All blocks have not the same number of accesses. The worst case scenario would be a unique access per block. We measure

this option in table 3.3 (see Accessed only one time row). When we calculated the total number of blocks accessed more that one time in Seq. A, we note that total blocks accessed more than one time is greater than accessed only one (see Accessed blocks gt 1 row).

	Seq. A		Seq. B	
Total Blocks	254,235		254,235	
Accessed Blocks	24,541	(9.6%)	599	(0.3%)
No accessed Blocks	229,694	(90.3%)	253,636	(99.7%)
Accessed only one time	1,996	(0.8%)	63	(0.03%)
Accessed blocks gt 1	22,545	(99.2%)	536	(99.97%)

Tabla 3.3 Blocks accessed from sequence A and B.

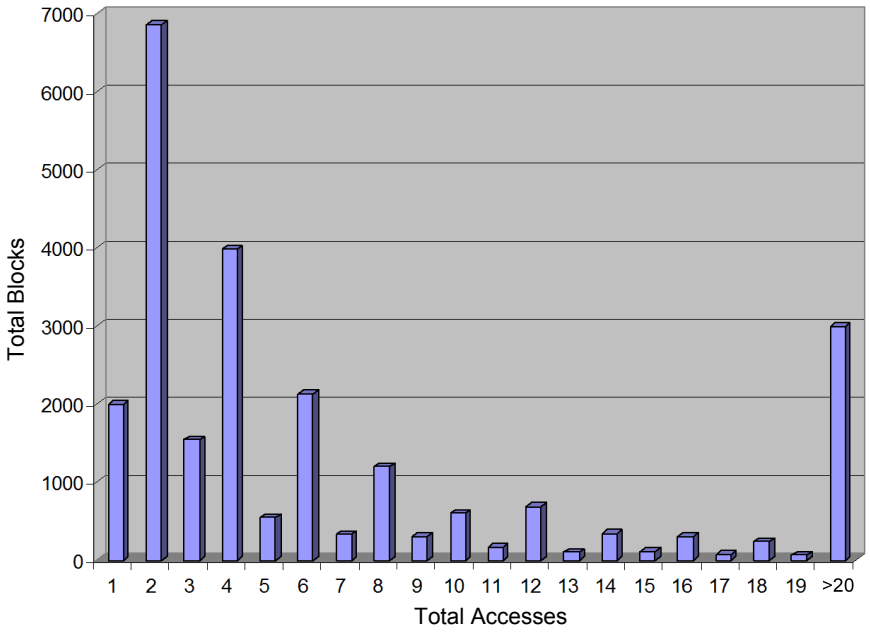


Fig. 3.4 Blocks grouped by total accesses executing Sequence A.

3.5 Conclusions

The article shows the susceptibility of improving applications that require accessing small parts of huge data files. The selection of the rightmost location of the data and the use of caching may have a clear impact on their performance. This is the case

of BWT-based alignment methods. Using previous BWT search code we produce a faster new version. To apply this solution is necessary a quick file transfer. We tested several strategies and measure downloading times. Transfer time may vary depending on infrastructure conditions (see section 3.4.2) and this will alter search total time.

By analyzing access pattern in intermediate file we can highlight that only a reduced group of blocks are used. At that time we suggest direct access to remote file. This choice was more expensive that previous ones using full file downloads, but a new system based in temporal and spatial locality references is suggested. A new layer above GFAL can reduce total access time and data transfer needs. This architecture will provide a new two level cache memory. Next step will be tuned up parameters and development of mock-ups to evaluate the proposed architecture.

Capítulo 4

Detecting Events in Streaming Multimedia with Big Data Techniques

Título:	*Detecting events in streaming multimedia with Big Data techniques*
Autores:	*José Herrera y Germán Moltó*
Publicación:	*24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), IEEE , 2016*
Licencia:	*©2016 IEEE Reprinted, with permission, from J.Herrera and G. Moltó, "Detecting events in streaming multimedia with Big Data techniques", Feb 2016. Autorización en Anexo A pág.143*
Referencia:	*[106] - https://ieeexplore.ieee.org/abstract/document/7445354*

El segundo artículo [106] estudia la detección de eventos en los tratamientos de grandes volúmenes de datos multimedia. Los datos tabulares y los datos multimedia tienen diferentes tratamientos dentro de las herramientas de Big Data, suponiendo un desafío diseñar un proceso capaz de tratar grandes volúmenes de datos multimedia de la misma forma que se utilizan los datos textuales o tabulares. La prueba de concepto (PoC) que se realiza, se centra en la utilización de una infraestructura de Big Data desarrollada en Apache Hadoop que ejecuta un proceso dividido en seis etapas.

- Etapa 1. **Captura de audio** durante intervalos de tiempo establecidos procedente de fuentes libres. Esta etapa tiene como objetivo la obtención del audio digitalizado, la generación de los ficheros necesarios para su tratamiento posterior utilizando ffmpeg [14] y su almacenamiento en el sistema de fichero HDFS.

- Etapa 2. **Normalización de los datos**. Cada uno de los ficheros generados en la etapa anterior es normalizado para que correspondan con los parámetros de audio establecidos previamente, especialmente los niveles de volumen. Este paso también sirve para adaptar los formatos de los datos capturados a un estándar (PCM, 16-bits, 44.100Hz) tratable por los procesos que se realizaran en las siguientes etapas.

- Etapa 3. **Diarización**. Es la separación, segmentación y clusterización de los participantes en el fichero de audio en función de las características del habla. Para la realización de esta tarea se empleó LIUM_Spkdiarization [230] como herramienta de código abierto.

- Etapa 4. **Reconocimiento**. Este paso se centra en el reconocimiento del habla de las porciones diarizadas en el paso anterior. Utilizando la librería Sphinx [229] de código abierto se permite, además, el reconocimiento en varios idiomas. Después de esta etapa se logra un flujo de palabras con una indicación de tiempo de aparición.

- Etapa 5. **Construcción del conocimiento**. Utilizando el texto reconocido y su secuencia de tiempo, se procesa en busca de palabras o grupos de palabras previamente prefijados, almacenando la información de detección para posteriores tratamientos.

- Etapa 6. **Visualización de resultados**. Los resultados de la construcción del conocimiento se centralizan en un único nodo y generan un gráfico que muestra las acumulaciones de las palabras utilizadas en los períodos de tiempo tratados.

Se revela que una de las tareas más importantes del proceso es la selección de un tamaño de bloque adecuado, parámetro que es primordial en la utilización dentro del sistema de almacenamiento de ficheros HDFS. Tanto el tamaño físico como el lógico resultan claves para la obtención de buenos resultados.

En las conclusiones se menciona la posibilidad de tratar los contenidos multimedia mediante infraestructuras diseñadas con el paradigma MapReduce. Después del tratamiento de los datos multimedia se pueden derivar numerosos conocimientos, entre los que se ha presentado la obtención de un gráfico para su exploración visual.

Abstract

The massive amount of multimedia information currently available through the Internet demands efficient techniques to extract knowledge from Big Data. In this work, we propose an architecture to capture, process, analyse and visualize data coming from multiple streaming multimedia TV stations and radio stations. For that, we rely on the Hadoop framework available within the IBM InfoSphere BigInsights platform. We create a workflow to automate the different stages that range from Automatic Speech Recognition using open-source tools to visualization by means of the R framework. We emphasize techniques such as diarization and the optimization of the number of Hadoop nodes, provisioned from Cloud infrastructures, to deliver enhanced performance. The results show that it is possible to automate knowledge extraction from multimedia data running on virtualized infrastructures by means of Big Data techniques.

4.1 Introduction

The unprecedented improvements in both computer networks and computing capacity of devices have paved the way for multimedia information to be widely consumed across the world. Nowadays, audio and video documents are accessed from all kinds of devices, tablets, mobiles [31], wearable gadgets or desktop computers. Current Big Data coming from multimedia sources across the internet demands modern techniques to efficiently process and extract knowledge from that data.

In this paper, we propose a workflow to process multimedia data coming from TV and radio stations that provide live streams through the Internet. Data is sampled and stored in a distributed computing cluster so that the audio information can be efficiently processed in parallel to extract specific knowledge in a reasonable amount of time. These knowledge includes: i) detecting the popularity of a certain term across different media in a specific range of time; and ii) summarise the most used terms in a certain broadcast emission, among others. This approach has many applications in different professional fields which include, but are not limited to, journalists, to obtain digital press summaries and linguistics, to research on the evolution of certain terms in multimedia data along the time.

After the introduction, the remainder of the paper is structured as follows. First, section 4.2 identify some works related to this one. Next, section 4.3 addresses the way we process such big volume of data, described as a workflow to capture and process streaming multimedia coming from different sources together with the different

tools employed to perform the data analysis. Later, section 4.4 presents and discusses the main results obtained after the execution of several case studies that assess the effectiveness of the proposed approach. Then, section 4.5 discusses the obtained results. Finally, section 4.6 summarizes the main contributions of the paper and points to future work.

4.2 Related work

There are other works in the literature that aim at analysing multimedia data using Big Data Techniques. In [138],[137] the authors propose a system to perform the parallel transcoding of video files on a Hadoop [235] cluster.

We can also find examples in stream processing. In [190] the authors describe the *split step* concept to find boundaries in streams a major problem in multimedia processing. As another example of video processing, the work by Schmidt et al [212] uses key frames (independent encoded frames that do not depend on other video data to be decoded) to split a media byte-stream into parts. In [96], examples of diarization and video processing are explained. They focus on a Python-based framework that automatically maps computation onto parallel platforms. Data mining techniques were used in [234] for video event detection. For audio based works, the work by Ahnn [2] also used voice recognition comparing different Hadoop clusters.

In this paper, we combine: i) data capturing via open-source libraries from actual streaming radio stations; ii) data pre-processing using diarization techniques to properly separate audio among speakers; iii) data storage, on the Hadoop Distributed File System of clusters provisioned from on-premises Clouds; iv) data analysis via different Map/Reduce jobs that extract knowledge from the raw data and, finally, v) data visualization, using the R framework to visually depict the extracted knowledge. The resulting workflow provides the foundation of a system to produce visual representations of aggregated information from different multimedia streaming sources.

4.3 Proposed modeling and workflow

This section introduces a six step model for batch broadcast processing based on Hadoop. Hadoop provides a cluster-based distributed computing framework based on the *MapReduce* [45] programming model and a distributed file system known as HDFS (Hadoop Distributed File System). A Hadoop cluster consists of a front-end node, which distributes the tasks, and a set of worker nodes, which actually execute the jobs.

- Step 1. **Capture Source**. The first step is to capture the multimedia data from live streams. For that, shell scripts that use *ffmpeg* to capture video from TV live streams and audio for radio stations are employed. This results in Macromedia Flash Video (H.264 video and MPEG-4 AAC audio) format file for each TV station and a MP3 (MPEG 1.0 Layer 3) format file for each radio station.

- Step 2. **Data Normalization**. To improve the speed of the next steps, we extract and convert all binary data in a Hadoop sequence file that contains only the data that will be used in subsequent phases. The output from Step 1 is filtered, normalized and converted into WAV file format (44.100Hz, 16-bit, PCM). After that, we create a record level compression Hadoop sequence file as an optimized way to process a large number of small files, which is known to introduce a significant overhead for HDFS [27].

- Step 3. **Diarization**. To increase data parallelism we diarize all source sound files and split audio streams into segments that will be efficient to process in next steps. Diarization stands for speaker segmentation, or the ability to partition an input audio stream according to the identity of each speaker. For this process, we have relied on LIUM_Spkdiarization [205]. The output of this step is another record level compression Hadoop sequence file with several small files that result from the diarization process.

- Step 4. **Speech Recognition**. The output files from the diarization step are the input to the speech recognition process, where audio data is converted to text data using CMU Sphinx 4 [229]. Notice that these files are efficiently processed in parallel on different nodes of the Hadoop cluster. Therefore, the workload can be efficiently tackled by increasing the capacity (in terms of the number of nodes) of the Hadoop cluster.

- Step 5. **Knowledge Building**. Using the text information produced in the previous step we derive some knowledge. For example, looking for a word or group of words, summarize the total word occurrences or detect specific events are possible developments (think about a high occurrence of the word *fire* in the recognised speech for TV station in a certain geographical area as a proxy for a real fire in that area). This step is also run in the Hadoop cluster using the Hadoop sequence files from the previous step.

- Step 6. **Show Outcomes**. This step uses R [68] as the visualization package to produce high quality charts to be displayed for end user. This step runs in the

master node of the Hadoop cluster and graphically visualizes knowledge extracted in the previous step.

Concerning the Hadoop distribution, we have used IBM InfoSphere BigInsights version 3.1 [34]. The computational nodes have been dynamically allocated from an on-premises Cloud managed by OpenNebula 4.8. The proof-of-concept involves three virtual nodes with 2 vCPUs and 2 GB of RAM (except the front-end node, with 4 GB of RAM). Data was stored on HDFS in the cluster. The capacities of the underlying physical machines on which the VMs where running are dual 4 core Xeon E5620@2.40GHz with 16 GB RAM. Hadoop clusters on Virtual Machines are deployed via EC3 [21] to rapidly deploy test environments logically isolated from other experiments being executed on the underlying production infrastructure.

4.4 Configuration and measuring process

The speech recognition engine requires an Acoustic and a Language Model to properly recognise TV and radio broadcasts, together with a recognisable dataset. For the experiments, a test model was configured using the Voxforge Acoustic and Language Model [166]. A dataset, replacing step 1 and 2, was created joining small WAV files from the spanish Voxforge speech corpus, also in [166], creating a continuous one hour file of audio (464 MB). This allows the study to focus on the Big Data techniques to process multimedia data rather than on the fine-tuning of the algorithms for proper audio recognitions, which lies out of the scope of this paper.

4.4.1 Setting block types

The main steps were grouped in three sub-workflows. The first workflow (WK1) groups step 1 and step 2, including data acquisition and data normalization. The second workflow (WK2) groups diarization (step 3) and speech recognition (step 4) and, finally, the third workflow (WK3), which includes step 5 and step 6, produces the results to be displayed for the user. Only WK2 and WK3 require jobs based on Hadoop and we have relied on *Apache Oozie* as the workflow scheduler to enact the different tasks of the workflow.

To further configure the cluster for data processing, we used two properties:

- Physical Block Size. Using the *dfs.block.size* parameter to control the Hadoop sequence file block size when the file is first stored in HDFS. This value can not be changed after file creation.

- Logical Block Size. Using *mapred.max.split.size* we can define the total number of jobs being concurrently executed in the Hadoop nodes. This value can be changed in the Oozie job when the execution of the workflow starts.

According to our early experiments, there are no noticeable time differences when specifying the size of the block via physical blocks or logical blocks. Thus, we only used logical blocks for testing purpose in this paper. By setting the block size we can obtain the total number of blocks (see Table 4.1) which results in the total number of jobs to be run in the Hadoop cluster.

Jobs/Block Size	32 MB	64 MB	128 MB	256 MB
Step 3	15	8	4	2
Step 4	57	29	14	7

Tabla 4.1 Total number of blocks created setting the logical block size.

In fact, if we use a block size of 32 MB in step 3 (diarization phase) for a one-hour sequence file then we will have to execute 15 jobs in parallel. If we set a 256 MB block size, then only two jobs will be required to run in the Hadoop cluster concurrently. In the same way, using a block size of 32 MB in step 4 will require running 57 jobs. By obtaining the total number of blocks created for a certain dataset we can calculate the speedup in the infrastructure by modifying the total number of nodes. Figure 4.1 shows that the best result for step 4 is obtained with a 32 MB block size and 57 nodes. Introducing additional nodes in the cluster does not increase the speedup because Hadoop processes a block only on a node. Therefore, the block size determines the number of blocks to be processed and, therefore, sets the maximum number of tasks to be executed in parallel which results in an inherent limit on the scalability of the executions for a given dataset on a cluster with a given number of nodes.

4.4.2 Execution results

Once developed, we tested all the workflows in the Hadoop cluster for different block sizes in several scenarios. Four measures were taken for step 4. First, the number of jobs executed for different block sizes, which is the same as in Table 4.1. Second, Table 4.2 displays the accumulated total time (Compute time) for the running jobs. The total execution time of jobs running out of 32 MB block sizes (like a sequential execution) require longer time than the total execution of jobs created with 128 MB block size. Increasing block size from 128 MB to 256 MB introduces a slight increase

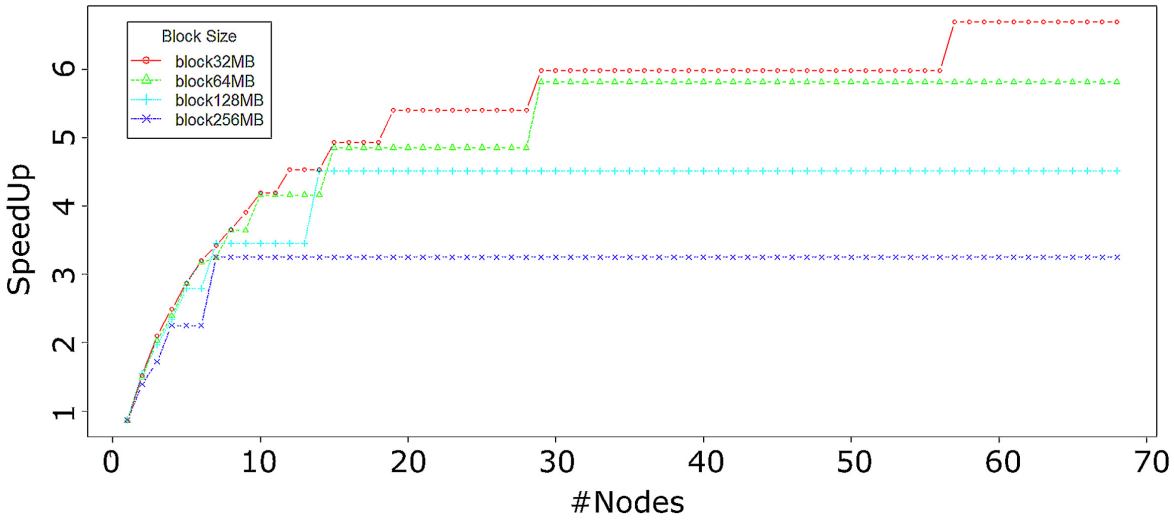


Fig. 4.1 Speedup simulation for step 4 (speech recognition).

in the total execution time. Finally, also in Table 4.2, the total *Reduce* time for jobs increases with the size of the block. Notice that this refers to the *Reduce* phase of the MapReduce execution for step 4.

Block Size	32 MB	64 MB	128 MB	256 MB
Accumulated time (sec)	52,448	41,323	39,839	39,886
Reduce time (sec)	4,982	5,053	5,350	5,652

Tabla 4.2 Execution results for step 4 using differents block sizes.

Data obtained from WK2 is used by WK3. Three knowledge extraction procedures were coded for WK3, named WK3-1 to WK3-3. First, WK3-1 processes the occurrences of words within a single day across multiple media, creating a word cloud where the size of the word grows with the number of occurrences. Secondly, WK3-2 shows average appearances in a two-axis graphic. Finally, WK3-3 represents data across several years of simulated data in a heat map (see Figure 4.2) using several sequence files created on previous steps. This represents the total occurrences of two words in the synthetic dataset generated. This kind of representation is easy to understand, and enables the user to conclude that certain words are more often used in a certain time frame (think of the use of words like *sun/hot* or *beach*, for example, during the warmer months of the year).

For Table 4.3, we tested the execution with different block sizes in jobs configured to run only one *Map* process and only one *Reduce* process. Using larger block sizes

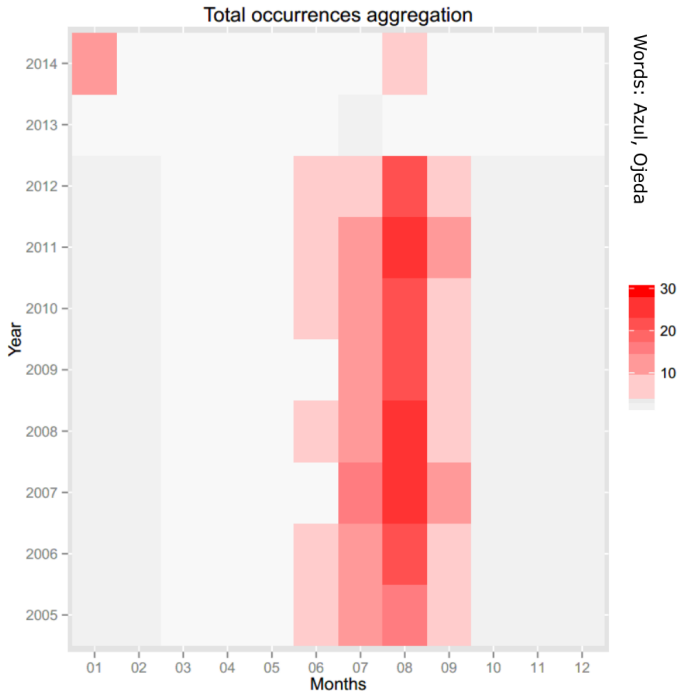


Fig. 4.2 Heat map interpretation of information created (WK3-3 plot).

yields decreased execution times, although that limits the degree of parallelism that can be achieved. Therefore, to reduce the overhead when processing many small blocks one should configure Hadoop to handle larger blocks.

	32MB	64MB	128MB	256MB
Step2 Map	2,051	2,067	1,988	1,797
Step2 Reduce	377	436	558	573
Step3 Map	42,448	41,323	39,839	39,886
Step3 Reduce	4,982	5,053	5,350	5,652
WK2 Total	49,858	48,879	47,735	47,908

Tabla 4.3 Average execution times for single Map/Reduce jobs in sec.

It is very interesting to analyse the behaviour of the *Reduce* phase when the block size is increased. In the developed workflows, the *Reduce* phase is always a single task. Keep in mind that the *Map* jobs are executed across the Hadoop cluster while the single *Reduce* task runs in just one node. After receiving data portions to process from the *Map* phase, the fragments were processed in *Reduce* to obtain step results. The processing time for the *Reduce* phase decreased with the number of blocks, i.e. when the number of *Map* jobs increases. To prove this point, we ran the same job using the

total *Map* jobs shown in Table 4.1 and a single *Reduce* task. The results draw the same conclusion, *Reduce* jobs spend less time with reduced size blocks, or what is the same, greater number of *Map* jobs (see Table 4.4).

	32MB	64MB	128MB	256MB
Avg. Map Jobs	744	1,424	2,845	5,698
Reduce job	4,982	5,053	5,350	5,652

Tabla 4.4 Total time in sec. for execution.

4.4.3 Main memory and file sizes requirements

After defining data structures and processing workflow, we should forecast the long-term amount of space required. We have to consider two main aspects for evaluating the storage requirements. First, the total storage requirement for Hadoop input/output sequence files must be measured. Capturing streaming for a year, using a replication factor of two for HDFS (where each block is replicated twice across the Hadoop cluster), we estimated a total size of 31 TB for each captured station. Second, the *NameNode* main memory requirements to store the Hadoop objects. The *NameNode* is a component in a Hadoop cluster that establishes the correspondence between the data blocks and physical nodes where files are stored. Consequently, it stores information about all objects (files, directories and blocks) and its main task is to serve metadata retrieval requests. In Table 4.5 we employ the size evaluation made in [220] to estimate the RAM required for the *NameNode* to store the block information after capturing ten years of data using a block size of 32 MB and file length of one hour. The size estimated is 14.21GB of main memory. If we increase the physical block size from 32 MB to 128 MB, the total main memory size required reduces to 3.55 GB due mainly to the amount blocks information that must be stored. Notice that this value does not include the massive amount of required storage space for the data files, which would be stored in HDFS.

4.5 Discussion

In a first approach we captured one hour data in a single Hadoop sequence file. All multimedia streams were encapsulated in a file that contains sixty minutes for each station. A total of 600 minutes stream data for ten stations was stored.

	#objects	Object Size (bytes) [220]	GB in MM
Files	70.080	250	0.16
Directories	44,177	290	0.11
Blocks	4,064,640	368	13.93
Total	4,178,897	–	14.21

Tabla 4.5 Ten years capture Main Memory (MM) requirements.

To create the drawings we used WK3-1 to WK3-3 jobs. Every workflow has different time scope. WK3-1 only processes a single one hour sequence capture file. The scope of WK3-2 spans captured data from a whole year. This represents 8,760 processed files. Finally, WK3-3 processes data captured over a ten year period, accessing a total of 87,600 files. Table 4.6 shows execution times (real for WK3-1 and WK3-2 but estimated for WK3-3) using the same test dataset replicated as many times as necessary.

According to the results obtained through the experiments, it is not a good solution to save WK2 outcomes in small files of one hour. Remember that the output of WK2 is the input for WK3. Therefore, for short range jobs (WK3-1), it is appropriate to access a reduced number of files. However, accessing a large number of files, as we do in WK3-2 and WK3-3, requires pre-processing files to assemble them together into bigger files in order to increase performance.

Workflow	Plot scope	Execution time	# Files accessed
WK3-1	1 hour	30 sec.	1
WK3-2	1 year	32 min.	8,760
WK3-3	10 years	6 hours (estimated)	87,600

Tabla 4.6 Earlier WK3 execution times. Worst approach.

Notice that evaluating the performance of the system in terms of speed is not the only measure used to determine system effectiveness, since other parameters can be considered. Section 4.4.1 indicated that no differences were found in execution time using physical or logical blocks, but further assessment showed us that other parameters can be properly tuned. In particular, to prevent excessive usage memory usage for the without *NameNode* service, virtual blocks should be employed. Reducing the virtual block size increases overload due to data transmission and block management, and it also increases total job accumulated time (see table 4.2), but the global MapReduce job completes in a shorter time due to parallelization. Therefore, we can conclude that using logical blocks is a better choice than using physical blocks.

Empirically, testing various block sizes gives us very valuable information. With a small block size more Map tasks can progress in parallel (assuming enough available resources in the Hadoop cluster) thus increasing the utilization of the nodes of the Hadoop cluster. However, in this case, boundaries were set by the number of segments created in the diarization process (step 3) and the total blocks generated by Hadoop (depending on the virtual block size). On the other hand, with large block sizes, compute time is lower, but parallelization is worse. This means that, for this particular problem, lower block size achieves better performance.

4.6 Conclusions and future works

This paper has presented a workflow-based approach and a proof-of-concept implementation of a platform to extract knowledge from streaming multimedia stations broadcasting through the Internet. The approach combines data capturing, data processing, data storage, data analysis and data representation techniques. A distributed processing model based on Hadoop has been employed to efficiently process in parallel the obtained datasets to perform different analysis to provide a graphical representation of aggregated information.

For that, we proposed a six step workflow for multimedia streaming processing using MapReduce programming model. Then, we provided an implementation of the different steps using open-source state-of-the-art tools. Understanding block management and analyzing block sizes in Hadoop framework has been essential for the efficient usage of hardware resources. Block types (physical or logical) and block sizes should be fine-tuned for optimal executions.

Extensive experimental results have demonstrated that the proposed workflow is able to work in the parallel *MapReduce* framework, thus reducing processing time on Hadoop virtual clusters which are provisioned from on-premises Cloud infrastructures.

Future works include ensuring that the workflow model performs efficiently also for event detection in video streaming. We also plan to migrate the implementation of the workflow to Apache Storm platform and/or in Apache Spark in order to compare different implementation approaches, with a special emphasis on real-time data processing.

Acknowledgment

The authors would like to thank the “Ministerio de Economía y Competividad” for the CLUVIEM project with reference number TIN2013-44390-R.

Capítulo 5

Logotype Detection in Streaming Multimedia Using Apache Storm

Título:	*Logotype detection in streaming multimedia using Apache Storm*
Autores:	*José Herrera, Germán Moltó y Atsuhiro Takasu*
Publicación:	*Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2017.*
Licencia:	*CSREA Press ©* *Autorizada su utilización ver Apéndice A pág.142*
Referencia:	*[108]* *https://csce.ucmss.com/cr/books/2017/LFS/CSREA2017/PDP3398.pdf*
Clasificación:	*Core B*

El tercer artículo [108] se centra en la implementación de la detección de logotipos en una fuente continua de imágenes (vídeo) mediante la utilización de una infraestructura de tratamiento de grandes volúmenes de datos en tiempo real. Con el uso de Apache Storm se procesa una fuente de datos multimedia y se obtienen resultados en tiempo real.

Con la utilización de tres diseños, que permiten la comparación entre procesamientos secuenciales y paralelos, se realiza una topología de 7 nodos (Figura 5.2). Las topologías se definen como los grafos de la organización de componentes. Los grafos acíclicos dirigidos (DAG) se componen de dos tipos de elementos: *Spout* (fuentes de datos *stream*)

y *bolt* (paso de procesamiento de datos). Con la combinación de varios elementos se logra un tratamiento en tiempo real de los datos contenidos en un vídeo capturado de una televisión comercial. Los nodos utilizados son:

1. **VideoSpout**. *Spout* o fuente de datos. Es el encargado de la ingesta de un flujo continuo de información que en este caso es un *stream* de vídeo.

2. **PresProBolt**. Es un paso previo consistente en la conversión del vídeo procedente del VideoSpout a unos parámetros estándar como color, tamaño y resolución, y se realizan las porciones necesarias para su procesado (segmentación de imagen).

3. **CartesianBolt**. En este paso se cruzan todas las porciones de los vídeos y los logotipos que deben localizarse.

4. **FindImageBolt**. Los pares imagen-logotipo son procesados utilizando los algoritmos SIFT y RANSAC proporcionados en la librería OpenIMAJ [42].

5. **BrandOccCountBolt**. Este bolt hace una agregación de las apariciones de los diferentes logotipos procedentes de los *bolts* de procesamiento anteriores. Esta adición se realiza de una forma rápida mediante la utilización de un *Counter Bloom Filter* [16] evitando la contabilización de varias respuestas para un mismo bloque y los problemas de los retardos en el procesamiento debidos a errores o diferencias en las capacidades de los nodos que ejecutan los procesos.

6. **InstantCountBolt**. Es un *bolt* opcional que permite el almacenamiento de los datos de ejecución de forma secuencial.

7. **VideoGeneratorBolt**. Es un *bolt* opcional que permite generar un flujo de vídeo que incluye la información con los resultados del procesamiento. Todo ello con el fin de poder obtener información sobre el tratamiento que se está realizando internamente.

Además de diseñar e implementar una topología válida para la realización de un procesamiento de flujos de vídeo en directo, se analizan los parámetros y los ajustes necesarios para el correcto funcionamiento del prototipo. De esta forma se analizan, entre otros, el tamaño de bloque tratado en los *bolts*, el total de nodos y las limitaciones de la transferencia de datos entre *bolts*.

Como conclusiones podemos indicar que se logró un sistema para el tratamiento de flujos multimedia, mejorando el desempeño con la utilización de un *Counter Bloom Filter* y la necesidad de transferir los datos entre los distintos *bolts* para que el procesamiento sea más rápido.

Abstract

The massive amount of multimedia information currently available through the Internet requires efficient techniques to extract knowledge from data. Indeed, processing video information using real-time Big Data techniques is currently a challenge for the existing data processing frameworks. This paper proposes and evaluates different topologies for logotype detection in streaming multimedia implemented by means of open-source libraries and supported by the *Apache Storm* distributed real-time computation system. Different data movement strategies across the topologies are defined and evaluated. An experimental analysis that involves logotype detection in an excerpt of the Formula One Suzuka Japan Grand Prix 2015 is performed. The results indicate that logotype detection in videos can greatly benefit from the distributed computing capabilities offered by Apache Storm.

5.1 Introduction

Video Analytics is one of the areas that is a challenge for Big Data due to the use of highly unstructured data. Indeed, the use of video has dramatically increased during the last years within our information society. As an example, the statistics published by YouTube in [119] indicates that a third of the people on the Internet has a YouTube account, watching hundreds of millions of hours on that platform and uploading more than 400 hours of video every minute [127].

Indeed, video analytics for trademark management is one of the areas that can significantly benefit from Big Data processing. The total amount of logotypes worldwide is estimated to exceed 20 millions and the number trademarks raises at a rate of seven millions per year [51] according to data of World Intellectual Property Organization (WIPO). The ability to track the appearance of a brand in a video enables to quantify its level of exposure to the media for a certain event.

In this paper, we propose a topology based on *Apache Storm* to process large amounts of multimedia data from streaming video. This topology is applied in order to extract knowledge concerning the appearance of brand logos in the multimedia streams. We aim at processing this data on-the-fly, without previously storing the data, by using a distributed computing framework. This approach has many applications in different professional fields which include, but are not limited to, advertising companies to track the fulfilment of the advertising contracts in live sports events

After this introduction, the remainder of this paper is structured as follow. First, section 5.2 describes the related work in the area. Second, section 5.3 briefly introduces

the *Apache Storm* framework and its ability to process tuples in real-time, we also describe the libraries and algorithms employed to solve this challenging problem, together with the topologies implemented in order to address the case studies envisioned. Third, section 5.4 assesses the effectiveness of the proposed implementation through case studies and highlighting some useful steps to take into account for the topology implementation. Finally, section 5.5 summarizes the main contributions of this paper and points to future work.

5.2 Related work

Video processing is a computationally and data intensive task. Indeed, a second of high-definition video, in terms of amount of data, is equivalent to 2.000 pages of text, according to Anyika et al. [167]. Gandomi and Haider [88] discuss surveillance, retail and video indexing for common video analysis applications using Big Data techniques. They discuss two architectural approaches to video analytics: i) server-based architecture where multiple videos are captured and routed back to a server that carries out the video analytics and ii) edge-based architecture, where the video analytics are applied at the device that captures the video. Application fields are diverse. As an example, Gantz and Reinsel [89] propose improving military intelligence comparing correlation pattern in videos captured from drones.

Indeed pattern recognition is wide research area in which trademark detection in images has been previously addressed [126] [189]. Pattern recognition for logotype detection in video is also an area of study that aims at detecting logotypes [38], detecting and removing [238], and even detecting logotypes in low resolution video[39]. Also, new technologies are being applied to address this problem, such as using deep convolutional neural networks in the work by Iandola et al. [116], surpassing state-of-the-art accuracy on popular logo recognition datasets.

Hu et al. exposed in [113] that multimedia is one of the six types of Big Data applications and it involves overcoming the semantic gap of multimedia data. Indeed, there are other works in the literature that aim at analysing multimedia data using Big Data Techniques. The work by Weishan et al. in [241] focuses on stream processing with others purposes such as proposing a combination between batch processing and real-time processing. They state that effectiveness and efficiency requires using both off-line batch processing capabilities and on-line real-time in-memory processing. They use *Apache Storm* for glasses detection analyzing the detection results and the framework performance in [60]. Processing images for face detection is the proposal of Dong-Hyuck

et al. [118], where the Storm distributed framework is employed. However, no execution results are included in the paper.

The work by Zhang et al. [240] use similar techniques for image processing but the results shown are focused on the detection process rather than on the platform employed. Finally, the books by Allen, Pathirana and Jankowski [7] together with the book by Leibiusky, Eisbruch and Simonassi [155] on Apache Storm have been fundamental to underpin the foundations of this work.

As a final note, Kesidis et al. in [135] expose open challenges in trademark retrieval. Due to the number of trademark registered it is necessary to consider methodologies based on distributed systems that can scale gracefully. There is substantial variability in trademarks including geometric, monochrome, stylized or plain variations.

5.3 Materials and methods

This section describes the frameworks, libraries and methods employed to tackle the problem of brand logo detection on multimedia streaming data.

5.3.1 Apache Storm

Apache Storm [83] is an open source distributed real-time computation system that can process streams of data. A topology, or Storm application, consists of *Spouts*, *Bolts*, *tuples*, *streams* and *stream grouping* specifications for transfers. Spouts are sources of stream data that read tuples and emit them into the topology, i.e., to be processed by the bolts. Bolts process the received tuples performing different functions such as filtering, aggregation, join, etc. A tuple is a data model defined for a named list of values, and a field can be an object of any type. A Stream is a set of tuples in sequence. The grouping specification is the way that a group of objects is transferred to the next step in the processing carried out by a Storm topology. Finally, stream grouping specifies how topology sends data between two components. Spouts and bolts execute in parallel, as part of the topology, across one or more worker processes. Storm attempts to spread the load, i.e., tasks to be executed, evenly across the workers.

To understand parallelism in *Apache Storm* it is convenient to know that every Bolt may run in a different level of parallelism. These are the main components:

- *Workers*. They execute a subset of a specific topology running its own JVM. A topology runs across one or more workers. By default, every node has four workers.

- *Executors.* An executor is a thread of execution that is spawned by a worker process and run in the worker's JVM. It runs one or more tasks for the same bolt or spout.
- *Tasks.* A task performs the actual data processing. It runs in an executor's thread and shares this thread with other tasks. An executor is used by all tasks in the same Bolt or Spout in the same worker.

As a final note, *Apache Storm* can run several topologies at the same time providing a web user interface to control the running cluster. The most important attributes of this framework are: simple programming model, support for multiple languages, fault-tolerant, transactional, scalable, reliable and fast.

5.3.2 Image detection library

An open-source image detection library is used to find the logotypes in the images extracted from the videos. There are two main open-source free vision libraries to process images. On the one hand, OpenCV (Open Computer Vision) [33] is library written in C/C++ designed for computational efficiency and strongly focused on real-time applications. It is a well documented and widely used project. More than 500 functions are implemented in OpenCV to cover areas such as robotics, security, medical imaging or factory product inspection. It also includes a general-purpose Machine Learning Library (MLL) focused on statistical pattern recognition. On the other hand, OpenIMAJ (OPEN Intelligent Multimedia Analysis in Java) [42] is a Java library and tool for scalable multimedia content analysis and image indexing. It includes broad state-of-the-art computer vision techniques. The distribution is made using a modular set of *jar* files under a BSD-style license. The design and implementation keeps all the components modular to maximise code maintainability as well as to make the library easy to use.

Both libraries employ similar algorithms to detect images. With all these similarities, we selected OpenIMAJ to implement our code that use the Scale-Invariant Feature Transform (SIFT) [163] to extract some interest pixels from images and describe them. To find analogue pixels, we use Random Sample Consensus (RANSAC) [62] to fit a geometric model called an Affine Transform to the initial set of matches. After the pattern detection process, we obtain a number of matches or similarities. Closeness can be compared with a threshold. If the number is greater, we understand that we have a frame that contains the logotype. Notice that the goodness of the systems greatly depends on the goodness of the library.

5.3.3 Probabilistic structures operation

In order to prevent counting duplicate logotypes, a mechanism to discover if detection was made previously is required. One way to perform this is using the Java Sets available in the Java Development Kit (JDK) to store the previously processed frames. However, this procedure is not ideal when using a Storm topology due to constraints in the size of the set and the speed required to perform such detection. Instead, a Bloom filter [16] (BF) is a space-efficient probabilistic data structure. It is based on the use of hash functions to define data codification in a bit array. It is employed to test whether an element belongs to a set. Elements can be introduced but not removed. It is important to point out that false positives are possible, but false negatives are not. There exist some variations from the original Bloom Filter. One of the most interesting, for the purpose of this paper, is the Counter Bloom Filter [54] (CBF) which allows to delete information previously inserted in the Bloom filter.

In order to understand the benefits of the Bloom filter compared to the existing implementations of sets available in the JDK, Figure 5.1 shows the total execution time, in milliseconds, for different operations, comparing the CBF with respect to the aforementioned set implementations from the JDK.

It can be noticed in figure that CBF does not achieve the best time performance. However, CBF excels at memory consumption. If we compare the memory required by the data structures, as shown in Table 5.1, then we observe a significant memory reduction by the CBF compared to other implementations.

Items	Hash Set	Tree Set	Linked Hash Set	Bloom Filter
1,000	70	70	70	1
10,000	711	711	711	19
100,000	7,215	7,215	7,215	195
1,000,000	73,133	73,133	73,133	1,954

Tabla 5.1 Structure size to store data (in Kb).

Therefore we adopted CBF as the underlying data structure to detect duplicate logotypes during the video processing.

5.3.4 Proposed topologies

To provide a code implementation independent from the image processing library used, we developed a wrapper with a well defined interface which isolates the topology steps

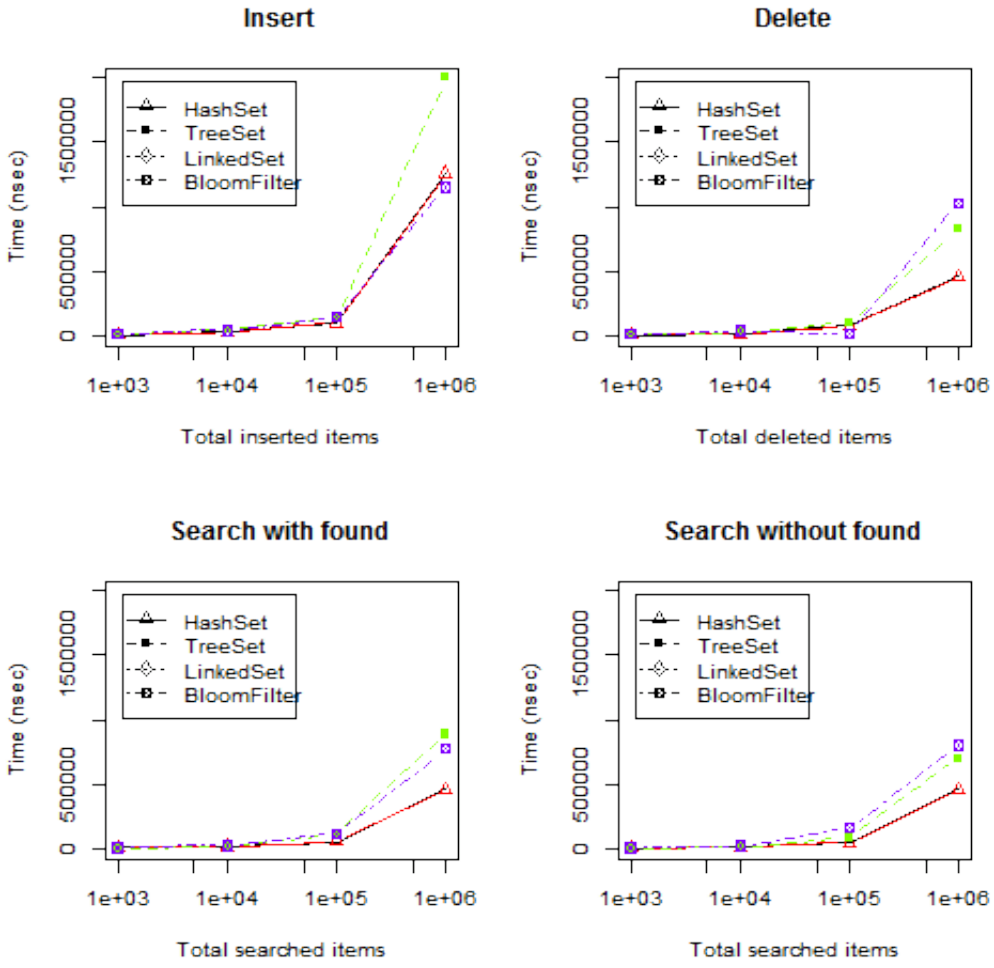


Fig. 5.1 Comparing Counter Bloom Filter execution with others set implementations in Java (all time expressed in nanoseconds).

from the library used. This way, only four methods are implemented: *load_frame()*, *load_logotype()*, *process()* and *get_total_points()*.

Concerning testing purposes, we developed three different code implementations. Topologies #1 and #2 are full *Apache Storm* topologies, while #3 code is a sequential execution that is used for comparison purposes. These are the implementations:

- Topology #1. **Transfer references in the tuples**. This first topology proposal is composed by only one spout process and several bolts, as explained in section 5.3.5. In this case, the spout reads a video file, separates it into individual images and stores them into a central node that will act as a repository of images. After saving the images, it sends image references to the next bolt creating a stream

of images with additional information (video origin and frame number) to be processed. The first goal in this proposal is to reduce the data exchange between the topology elements. Since only the image reference is transferred, bolts that use image bitmaps need to access this centralized node to retrieve the images. This way, data exchanges between topology nodes are clearly reduced at the expense of increasing the load of reads in the node with the repository of images.

- Topology #2. **Transfer images in the tuples**. To improve processing speed, we created a topology using the same directed acyclic graph but, in this case, the data transferred between topology elements includes transferring the images content between executors by inserting the image serialization bits into the tuples. Under these conditions, the Spout splits the video file, serializes the images using Java serialization and sends a tuple with the image serialization bits and the same additional information as in the previous topology. An image processing bolt, deserializes it, processes the image and serializes it again. Data exchange is increased as well as the processing time dedicated to serializing and desearializing the image data. However, no access to the image repository node is required in order to retrieve the images.

- Code #3. **Sequential Execution**. In order to compare the distributed Storm-based executions with a sequential execution, we developed a code that executes on a single processor (with multiple cores) and uses threads to obtain and process the video data.

5.3.5 Topology composition

Both topologies, #1 and #2, have the same directed acyclic graph, spout and bolts. We designed a five mandatory step topology adding two optional bolts to control topology operation.

1. **VideoSpout** (Compulsory Spout). This step creates a stream of data. In our case, this stream of tuples is a source of frames (images) or references that compose a video. In Topology #1 the stream only refers to the frames. In Topology #2, we create a stream of objects called *image_enhanced* that contain the serialized image bits and frame number. For testing proposal these streams are created from a video with MPEG format (ISO/IEC JTC1/SC29 WG11).

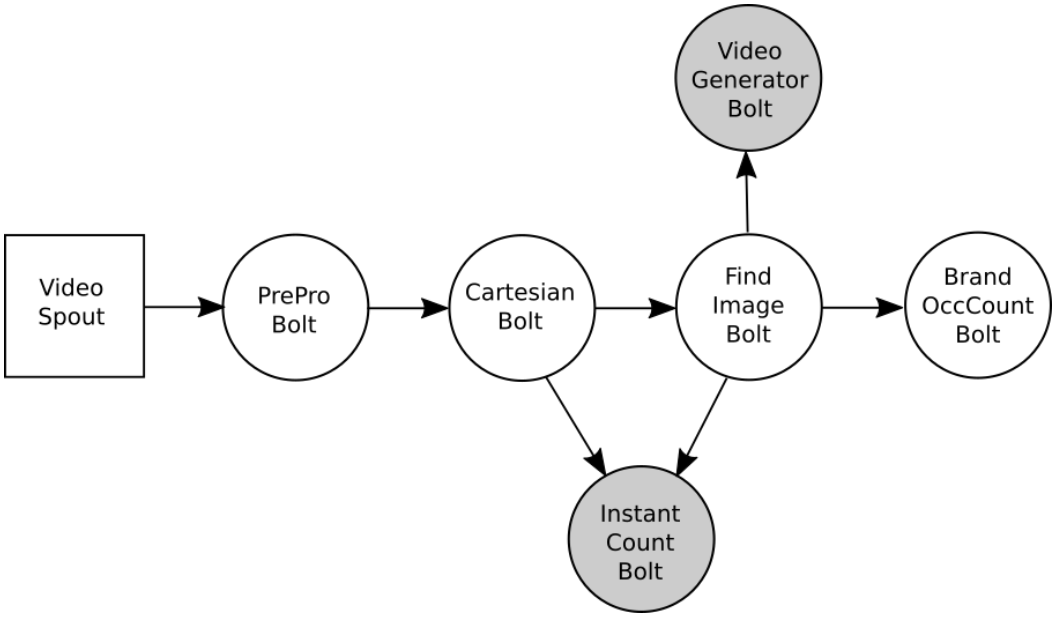


Fig. 5.2 Acyclic graph of implemented topologies #1 and #2.

2. **PreProBolt** (Compulsory Bolt). The pre-processing bolt is the first processing step in our topology. For example, image segmentation, black/white colour or border detection are processing options to improve the algorithm used. The key goal is to adapt and adjust the input frame to obtain the best results. Content analysis algorithms work better using images under certain rules. In this work, only a size reduction was made since the image size clearly dominates the total execution time. The final result of this bolt is an image properly adapted to the detection algorithm used.

3. **CartesianBolt** (Compulsory Bolt). In sport broadcasts, logotypes appear on different positions and with different designs. Corporate image manuals have available trademark images with alternative horizontal and vertical orientations, with different shape and necessary adjustments of colour, shape or lighting. Alternative designs can appear on a variety of places to increase the scope of advertising. For our purpose, this may hinder the ability to recognize the logotypes. To overcome this problem, we take a sample of each of the different designs. The sampling number is only conditioned to the abilities of the recognition algorithm. Making in this bolt a cartesian product of input tuples (source image) and logotypes samples, we create an output tuple stream of pair image-logotype. This significantly increases the number of recognitions at the expense of multiple logo detections.

4. **FindImageBolt** (Compulsory Bolt). In this part of the topology sequence, pairs image-logotype are delivered to this bolt to be processed. After getting both images, the content analysis algorithms are applied. Then, after the use of SIFT and RANSAC algorithms provided in OpenIMAJ library, it is obtained a degree of similarity that is compared with a global threshold parameter. The major computational complexity of this step is due to the recognition library.

5. **BrandOccCountBolt** (Compulsory Bolt). As a result, this bolt writes a line for each recognized trademark. Listing 5.1 shows the usual output that includes a timestamp, the file processed, the frame number and the logotype detected. Sometimes, specially in topology executions that involve heavy executions, some pairs of image-logotype are processed multiple times. This is because of losing the tuple delivery acknowledge due to very large processing times (timeouts in the topology) in the FindImageBolt step together with overloads in the hardware running the Storm workers. As a result, two positive detections could be received in this bolt. To solve this issue, we implemented a window scale control using a Counter Bloom Filter that avoids writing several times the same frame. Also, in case of delayed old frames, it filters out them using the same procedure.

```
1447806008922 japan_gp_15_split.mpg 00000053 pirelli
1447806011650 japan_gp_15_split.mpg 00000006 amd
1447806012235 japan_gp_15_split.mpg 00000054 pirelli
1447806013423 japan_gp_15_split.mpg 00000042 amd
```

Listing 5.1 Output unordered sample

6. **InstantCountBolt** (Optional Bolt). As an auxiliary process to describe the way the topology is working, this Bolt summarizes the video frames that are processed in the topology and the number of total positive logo recognitions during the last 10 seconds. It is not a required bolt but it helps us to understand the execution progress of the topology.

7. **VideoGeneratorBolt** (Optional Bolt). As in the previous case, this optional bolt is used to analyze the work made by the topology. Using the input from previous bolts, it creates a motion picture. With data processing results, existing brand and image processed, it joins all pictures. After that, we analyse the sequence to understand the process made and to debug problems and results from the *FindImageBolt* phase.

5.4 Configuration, testbed and measuring experiments

To evaluate the effectiveness of the system designed, we selected an excerpt of a pre-recorded video from the Formula One Suzuka Japan Grand Prix 2015. A total of 500 frames were selected from a bigger stream in which the frame size is 1280x720 pixels. Using a speed of 50 frames per second, the slice has a duration of 10 seconds.

We tested black and white image transformations on a sample dataset. However, no significant differences were obtained in the ability to recognize the logotypes. Therefore, all images were created, stored, transformed, serialized or deserialized used True Colour (24 bits) colour depth.

Images are scaled down for some tests maintaining the aspect ratio. In particular, 1.280x720 images are scaled down up to 500x281 pixels. In the figures, we only indicate the width values for the images, in pixels.

5.4.1 Experiment 1

For our first experiment, the Storm cluster is composed by 4 Virtual Machines (VMs) with 2 vCPUS (emulating the Intel Xeon E312xx processor), 15 GB of RAM and 30 GB of hard disk. All nodes are running Apache Storm except one that also runs Zookeeper [85] and a master node daemon (so-called internally Nimbus). The Apache Storm version used is 0.9.5 both for the development and execution environments, Zookeeper version 3.4.6 and Zeromq version 2.1.7.

Execution Results

The first test evaluated the data processing capabilities of the topologies. Figure 5.4 shows the time required for the execution of the three topologies varying the frame size, using a 500 frames stream (a ten seconds video). The worst topology was T. Data (Topology #1, see section 5.3.4), where only the references to the images are transferred between bolts. Sequential Execution #3 and Topology #2 (Transfer Image) have similar execution times. However, the sequential execution ran in one node with 2 cores and Topology #2 ran in the Apache Storm cluster (four nodes each with 2 cores).

Table 5.2 includes the results when the frame size for Topology #2 is changed. It evidences that for a 500 pixels frame size it only processes 1 fps. If the input stream is HQ 50 fps, we need to increase the size of actual test infrastructure (4 nodes) until 50, assuming that the problem scales linearly.

Fig. 5.3 Test set video frame sample.

	500	600	700	800	900	1,000	1,100
Speed	1	0.83	0.71	0.63	0.56	0.50	0.45

Tabla 5.2 Number of frames per second (fps) processed by the Topology #2 for different frame sizes (in pixels).

5.4.2 Experiment 2

The second experiment is designed to evaluate the scalability of the topology when the number of nodes is increased. A total of 10 nodes were configured in order to evaluate and compare time with previous experiments. The nodes have the same computing performance as those used in Experiment 1. The test set used has the very same frame size and total number of pixels as the one previously used. We limited the measurements to the Topology #2.

Execution Results

In Figure 5.5, we compare the execution of Topology #2 with the sequential execution (Sequential Execution Code #3) for different frame sizes and runtime with different number of nodes (4 and 10 nodes) for the Apache Storm cluster. We evaluated the execution time to process the dataset using a limited image range from 500 pixels to 1.100 pixels. No memory problems or limitations were detected in the 10 nodes execution.

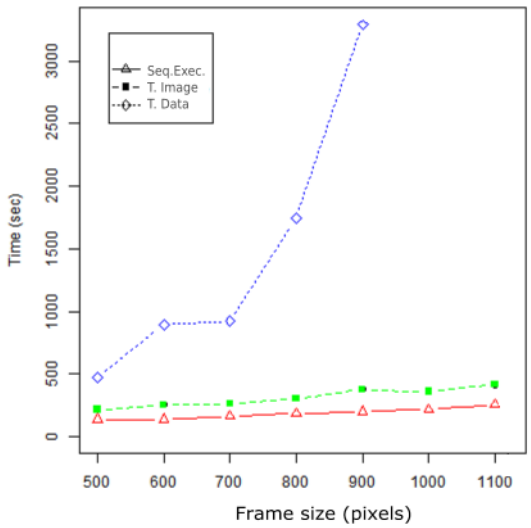


Fig. 5.4 Execution time of the different topologies.

As expected, the best results are obtained in a 10-nodes infrastructure. This indicates that the topology scales accurately. Also we observe that transfering data in tuple makes the topology less sensitive to changes in the frame size.

5.4.3 Discussion

It is important to identify the limiting factors that have a direct impact on the execution time of the topologies proposed.

1. **Video frame size**. One of the most limiting factors is the size of the problem. We need a high resolution video to find the largest possible number of trademarks. If the image size is too small, no logotypes can be detected since the resolution is not high enough to distinguish these logos. However, bigger video sizes increase the processing time.

2. **Total nodes**. A reduced number of nodes in the Storm cluster increase the execution time. To achieve a real-time logo identification procedure, a suitable number of nodes are required to process frames existing in every time period. As identified in Table 5.2, 1.4 secs. are required to process a single frame with 700 pixels resolution.

3. **Pre-processing and post-processing works**. Preprocessing is a big limiting factor. Converting the frame to black/white or modify frame size before recognition limits the system speed. At the same time, post-processing data mining

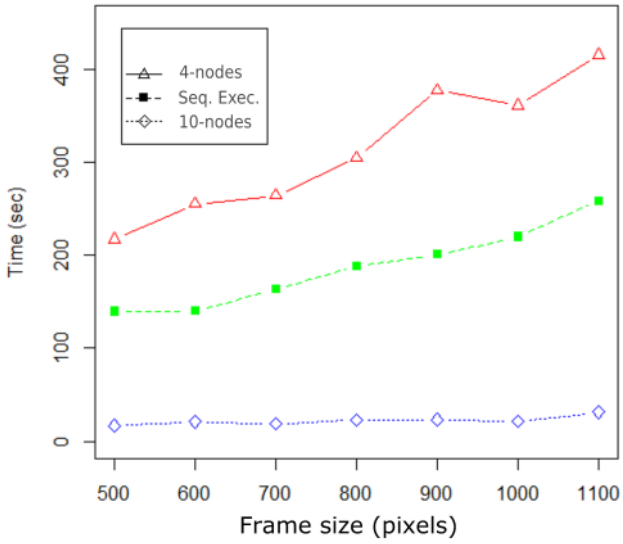


Fig. 5.5 Comparing #2 topology execution speed using ten nodes, four nodes and a sequential execution.

techniques, made after leach logo search, limits process output throughput. For that, we need a new bolt to ensure that logotypes are counted only one time for every frame.

4. **Transfer data limitations**. Other related factor is transferring data between bolts. Tuples were transferred between nodes to perform video processing. In one of our topologies only the image references were transferred. The pointers to real image node/position was moved between nodes. In the second topology, streams are composed by images inside an instance of a class prepared for this target. In all cases tuples are serialized and deserialized for each internode transfer. As a result, the second option reveals to be a better and faster approach.

5. **Total logotypes to search**. Tests and observation tell us that one image brand not only appears in one logotype. There exists different colors, shapes and sizes for each one. Perhaps, we want to identify two different logos in only one broadcast. To solve this difficulty, we create a group of logotypes associated to a brand that is detected at the same time but in different bolts.

6. **Total frames pending to be processed**. The last factor identified is the total number of frames that was in the topology pending of processing. This shows us the capacity to accept tuples and wait until they can be processed. This value is driven for the variable *max_spout_pending* in a topology. A very small value can

easily starve our topology and a sufficiently large value can overload the topology with a huge number of processing tuples to the extent of causing failures and replays.

Regarding system speed, transferring images in tuples is always faster that using a centralized image repository. Apache Storm lacks a distributed file system, as opposed to other Big Data frameworks such as Apache Hadoop with its HDFS (Hadoop Distributed File System). Also, Storm is not optimized for parallel file access. Indeed, topologies must be designed to prevent access to remote file systems. Apache Storm was an useful tool to process video streams. In fact, processing multiple video input was not be an obstacle to our developed topologies, achieving the same throughput than using single video inputs.

It is important to point out that the total number of nodes have to be increased to achieve a real-time execution of the topology. Using a non realistic linear estimation, around 200 nodes are estimated to be required to process a 500 pixels width frame and looking for two logotypes to obtain throughput comparable to near real-time processing. Taking into account this factor, using a faster library with improved algorithms may reduce the number of nodes required in cluster. State-of-the-art proposals, such as the use of deep learning algorithms, as Najafabadi et al. suggest in [181], can overcome the limitations and reduce the total execution time.

5.5 Conclusions and future works

This paper has focused on using the Apache Storm distributed data processing framework together with open-source image recognition libraries in order to identify logotypes in streaming multimedia information. We proposed two approaches of a seven step topology for multimedia streaming processing in which video information is adjusted to better match the requirements of the processing algorithms. Different implementations have been performed to evaluate different strategies concerning data movement across the components of the topology.

Several experiments have been carried out that resulted in the following conclusions: i) the usage of the Counter Bloom Filter significantly reduces the memory consumption for the process of identifying whether a logotype had already been detected; ii) image data should be transferred between bolts for better efficiency, as opposed to letting bolts retrieve the images before processing and iii) real-time processing of streaming video information cannot be achieved with modest size clusters.

The experimental results prove that the proposed topology is able to benefit from the distributed processing capabilities in the Apache Storm framework reducing the processing time and moving forward to real-time video processing.

Future works involve focusing on using alternative image recognition libraries and increase the total number of nodes used for job processing by using public Clouds in order to attempt achieving real-time video processing for logotype identification. Other streaming processing frameworks for Big Data such as *Apache Spark* [81] will be evaluated.

Acknowledgments

The research in this paper is supported by International Internship Program in National Institute of Informatics (NII) in Japan. GM would also like to thank the Spanish "Ministerio de Economía y Competitividad" for the project TIN2016-79951-R and for the project TIN2013-44390-R.

Capítulo 6

Towards Bio-inspired Auto-scaling Algorithms for Container Orchestration Platforms

Título:	*Towards bio-inspired auto-scaling algorithms for container orchestration platforms*
Autores:	*José Herrera y Germán Moltó*
Publicación:	*IEEE Access, vol 8 , pp 52139-52150, 2020*
Licencia:	*Creative Commons Attribution 4.0 International (CC BY 4.0)*
Referencia:	*[107] - https://ieeexplore.ieee.org/document/9036958*
Clasificación:	*JCR 2020. Comp. Science, Information Systems 23/155 (Q1)*

El cuarto artículo [107] trata de definir un sistema de autoescalado distribuido para contenedores software dentro de una plataforma de orquestación de contenedores como puede ser Kubernetes [160] o Apache Mesos [71]. El algoritmo empleado, basado en los autómatas celulares y en modelos bio-inspirados, parte de una correlación entre las acciones de escalado (horizontal: *scale-in*, *scale-out* o vertical *scale-up* y *scale-down*) y el ciclo de vida de las células biológicas. De esta forma se descubre una relación de similitud entre el escalado horizontal y vertical con las actividades del ciclo de vida celular.

Se realizan varios experimentos para valorar las estrategias alternativas y los valores límite de las acciones de escalado aplicables. Utilizando una carga sintética se confrontan las hipótesis con los datos obtenidos valorando las diferencias entre los valores esperados y los observados. La mejor propuesta es comparada con algunos

algoritmos de autoescalado disponibles actualmente en plataformas Cloud de uso común. El método de comparación utilizado es usar la contraposición de medidas comúnmente utilizadas (MAE, RMSE, EVS y R2S) y las distancias DTW obtenidas. El algoritmo *Dynamic Time Warping* (DTW) [207] permite conocer con mayor exactitud el ajuste de las diferentes alternativas basándonos en la comparación de la curva óptima y la respuesta durante la ejecución. DTW es una de las posibles opciones para tratar de conocer las diferencias entre series temporales que pueden variar en velocidad. De esta forma, proporciona la distancia del mejor ajuste que se pueda dar entre dos series temporales.

La fórmula utilizada para el cálculo de las acciones a tomar permite que cada uno de los contenedores exhiba un comportamiento diferenciado en función de las características que se desee. La actuación de cada uno de los contenedores puede ser adaptada rápidamente y de forma autónoma, en base a una función diseñada para obtener el mayor ajuste del sistema ante variaciones del volumen de datos.

Como conclusión, se obtiene un sistema de autoescalado distribuido mediante algoritmos de valoración avanzados que tiene buen comportamiento en las cargas de trabajo que son atípicas. El modelo requiere un ajuste óptimo para adecuarlo a las características que sean prioritarias (coste, precio, tiempo de respuesta, etc). De hecho, exhibe un comportamiento de sobreaprovisionamiento que puede llegar a suponer un coste a tener en cuenta cuando se produzca durante un período de tiempo prolongado o cuando el valor de las operaciones para proveer los contenedores sea elevado. La detección del número total de contenedores en ejecución es una tarea compleja de forma distribuida lo que supone un problema fijar una limitación máxima y mínima de los elementos en ejecución. Como conclusión se puede valorar que se producen mejores respuestas que con otros algoritmos cuando las cargas no siguen un patrón temporal, cuando las respuestas no son predecibles o bien cuando se producen valores atípicos.

Abstract

The wide adoption of microservices architectures has introduced an unprecedented granularisation of computing that requires the coordinated execution of multiple containers with diverse lifetimes and with potentially different auto-scaling requirements. These applications are managed by means of container orchestration platforms and existing centralised approaches for auto-scaling face challenges when used for the timely adaptation of the elasticity required for the different application components. This paper studies the impact of integrating bio-inspired approaches for dynamic distributed

auto-scaling on container orchestration platforms. With a focus on running self-managed containers, we compare alternative configuration options for the container life cycle. The performance of the proposed models is validated through simulations subjected to both synthetic and real-world workloads. Also, multiple scaling options are assessed with the purpose of identifying exceptional cases and improvement areas. Furthermore, a nontraditional metric for scaling measurement is introduced to substitute classic analytical approaches. We found out connections for two related worlds (biological systems and software container elasticity procedures) and we open a new research area in software containers that features potential self-guided container elasticity activities.

6.1 Introduction

The widespread adoption of Linux containers, and in particular Docker [122], as a mechanism for convenient application delivery has paved the way in the last years for the surge of the microservices architectural pattern [50] in which monolithic applications coded in a single programming language can be broken down into multiple polyglot services exposing interfaces. These are typically delivered and executed as containers managed by a Container Orchestration Platform (COP) such as Kubernetes [160] or Apache Mesos [71]. A COP acts as a scheduler for the execution of container-based workloads and provides secure access management to the pool of shared computing resources which are typically delivered in the shape of a cluster of computing nodes.

Previous literature agrees on the performance advantages of applications running on containers when compared to other virtualization technologies (see for example Felter et al. [58]). Indeed, an application running in a container can deliver a similar performance to when executed directly on bare metal. In addition, server density can be higher because no extra Operating System services are executed and, therefore, more containers can be executed per machine. However, microservices applications lead to faster creation, operation and removal of computing entities (containers) when compared to using Virtual Machines. This imposes a serious challenge for auto-scaling where more adaptable, precise and capable systems are required to manage the elasticity of large-scale fleets of containers belonging to multiple application architectures with dynamic elasticity requirements.

Adapting computational systems to the dynamic workload has been widely studied in previous works based on virtual machines and containers (Qu et al. [198], Hoenisch et al. [110]). Indeed, auto-scaling systems are already available for certain platforms, but the requirements of emerging application architectures requires a review of the

methods used to perform auto-scaling in heterogeneous platforms of containers. In particular, it is important to understand the implications of current research in other areas, for the field of elasticity of container-based computing platforms.

One of these areas is bio-inspired algorithms, where lifelike biological evolution is used as an inspiration source to solve a complex problem in computer science by finding an optimal solution, and a set of meta heuristics that mimic a biological process are adopted, typically to address optimisation issues [15]. Coding a system using an algorithm based on natural living sometimes introduces a convenient approach to solve the challenge [132]. Addressing auto-scaling of containers by means of bio-inspired algorithms may produce a viable option to quickly respond to changing requirements or load peaks.

To this aim, this paper introduces a set of novel bio-inspired algorithms aimed at supporting auto-scaling in container-based computing platforms. After the introduction, the remainder of the paper is structured as follows. First, section 6.2 provides an overall overview of the related state of the art. Next, section 6.3 underpins the similarities and relations between auto-scaling and bio-inspired algorithms. Then, section 6.4 proposes two models for auto-scaling based on cell modeling. Later, section 6.5 incorporates a proposed model evaluation to assess options, parameters and performance. It also includes experiments using the best proposed model versus well-known algorithms using synthetic and real world workloads. Finally, section 6.6 summarizes the main achievements of the paper and concludes the paper with future work.

6.2 Related work

This section summarizes the related work in the area of auto-scaling focusing on container orchestration platforms and bio-inspired algorithms.

6.2.1 Auto-scaling

A key feature of cloud computing is elasticity (Muñoz-Escoí and Bernabéu-Aubán [179]), where applications can dynamically adjust the computing and storage resources. On the one hand, vertical elasticity involves increasing or decreasing the amount of computing and memory resources of a single computing entity (typically a virtual machine in a cloud provider). On the other hand, horizontal elasticity requires changing the architecture of the application to run on a distributed fleet of computing nodes

that can grow and shrink according to the values of certain metrics, such as the average CPU usage in the last t seconds across the available nodes.

If no human intervention is required to adjust those resources, then auto-scaling is taking place [162]. Well-known examples of auto-scaling services are ECS Service Auto Scaling [121], autoscaling groups of instances for Google Cloud [161] or Microsoft Azure Autoscale [32]. A rich taxonomy of auto-scaling systems is described in the work by Qu et al. [198] and in the work by Al-Dhuraibi et al. [6]. Dynamic vertical scaling has also been addressed in the past through the dynamic allocation of memory to Virtual Machines, as described in the work by Moltó et al. [176]. Even public Cloud providers are starting to provide this functionality to some extent, as is the case of Jelastic [128].

It is common to find centralised auto-scalers that have an overall view of the state of the platform and provide scaling methods, resource estimation or scaling timing. Other studies based on containerisation such as the proposal by Kukade and Kale [146] or the one proposed by Kan [131] also involve centralised systems. Autonomic resource provisioning has appeared recently as a rich research field. Some examples are explained in Calcavecchia et al. [22] proposing a complex architecture and requiring agents relations. Najjar et al. [182] proposes a double type of modules that run as agents while An et al. [11] focused on allocating network resources in dynamic environments. Chieu and Chan [29] describe a complex system with centralized data structures for coordination without any type of evaluation and Son and Sim [222] proposed agent-based testbeds focused on Server Level Agreements (SLAs) negotiation, price metric and time-slot utilities.

In an auto-scaling system, scaling decisions such as deploy or terminate a node in horizontal scaling are taken at a certain point of time and are typically followed by a *cool down* period in which no scaling actions are taken to avoid oscillations in the system. Therefore, scaling decisions are not continuous. In fact, the time between two consecutive scaling actions depends on the monitoring interval (slot time), and it is important to measure the optimal time distance between reconfiguration actions. On the one hand, a bigger distance between actions causes a worse adaptation of the system to the arising workloads. On the other hand, a lower time between scheduling actions may introduce an overload in the elasticity system. Some mixed methods are explored. For instance, Rad et al. [5] expose a vertical scaling procedure mixed with live container migration.

A new era of auto-scaling systems based on a wide variety of variables recently appeared. Examples of high-level metrics, such as request service time, appear in

the work by Kaur and Chana [134], whereas adjusting the estimation of service time is described in Jiang et al. [129]. This suggests that the system must be capable of processing event-based workload prediction, provisioning using new pricing models, energy and carbon-aware systems (environmentally friendly) and more that will appear in the next years, considering previous variables too (classic economic cost and/or system performance). Although this work is focused on the reactive auto-scaling model [177], it is important to analyse that different nodes could use different values to evaluate options for auto-scaling.

A biological proposal for autoscaling is included in Messias et al. [174] where they evaluate a combination of time series prediction with an evolutionary algorithm using a centralized auto-scale architecture that start and stop virtual machines in a cloud computing platform. The reconfiguration interval or slot time elapsed 1 h according to most cloud infrastructure providers minimum period of charge, at the time when the study was carried out.

Virtual machine and software containers exhibit different values for resource provisioning, as demonstrated by previous works in the literature. The difference between provisioning a Container or a Virtual Machine has been assessed in Hussain [115], in the work by Gupta et al. [99] or in the work by Rad et al. [199]. Indeed, containers are just processes executed by the underlying OS with some visibility restrictions and, therefore, do not involve the resource provisioning and boot up times required by VMs. Actually, it is very common to run containers (application delivery) on top of Virtual Machines (infrastructure provision).

6.2.2 Bio-inspired models and cell automaton

A cellular automaton is a mathematical model for a dynamic system composed of a set of cells that acquire different states or values. These states are altered from one moment to another in units of discrete time, i.e., that can be quantified with integer values at regular intervals. In this way this set of cells evolves according to a certain mathematical expression, which is sensitive to the states of neighbour cells, and is known as the local transition rule [233] [20]. Models based on individual agents allow personalised "learn", adaptation and reproduction [90].

Swarm intelligence is based on group behaviour of species like ants, bees, etc. These species have an intelligent behaviour without a centralized authority. Simple agents or bodies interacting individually with the environment or among them generate a global desired behaviour (see [48, Chapter 9]).

6.3 Relating bio-inspired models to auto-scaling

In this section, we analyse important underlying parallelisms and similarities between bio-inspired algorithms and auto-scaling actions. In both scenarios, decisions are taken at certain time intervals that affect the size of the population either by adding new individuals (i.e. deploying a compute node, which can be a container or a Virtual Machine) or killing individuals (i.e. shutting down a compute node).

The basic elements in a cellular automaton can be related to auto-scaling of containerised applications. A state set is possible both in cellular automata, and in auto-scaling systems. The initial configuration would be the same in both systems, and a transition function must be defined in both cases. Auto-scaling systems typically define a transition function for a centralised system. However, in cellular automata a transition function is defined for every single cell. By adopting the behaviour of cellular automata distributed auto-scaling can be performed without requiring a single centralised entity taking decisions about the scaling behaviour.

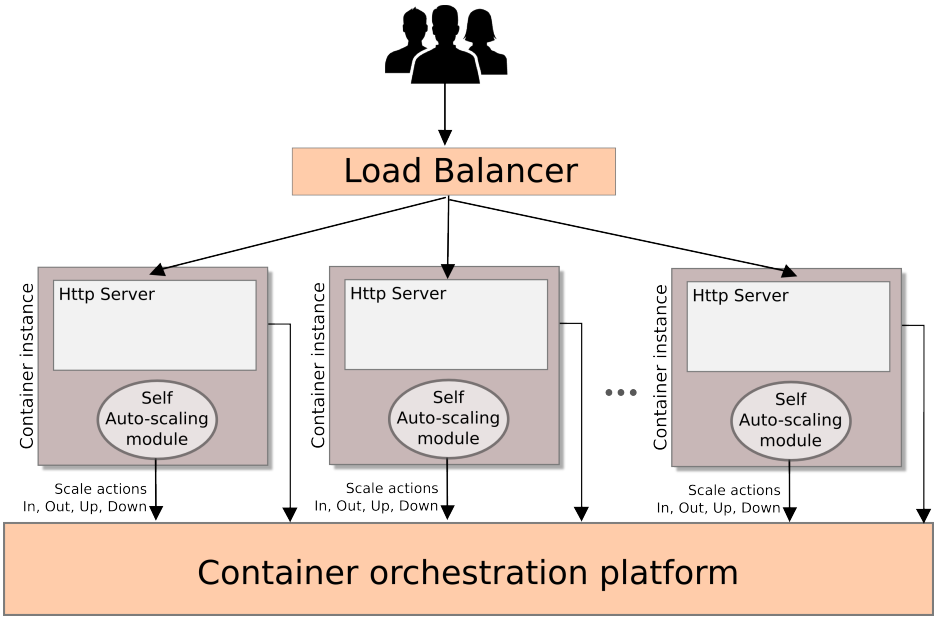


Fig. 6.1 Logical distributed architecture proposal.

Some extrapolated advantages are: evolutionary algorithms are intrinsically parallel. They are a representation of an independent entity operating autonomously and, therefore, actions are taken individually. They work especially well for solving problems whose space of potential solutions is large and non-linear. They do well in problems with a complex adaptive landscape, such as those in which the fitness function is

discontinuous, noisy, changes over time, or it has many local optima [52]. They have the ability to manipulate many parameters simultaneously and exhibit a reduced time lapse for decision making. Also, decision in time t is faster and easier to take than in complex algorithms.

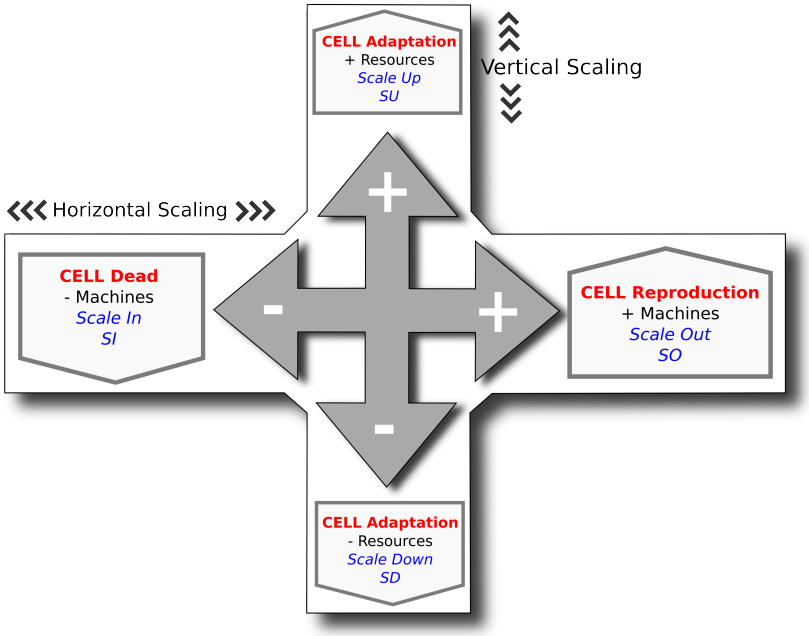


Fig. 6.2 Relations for scaling actions, cell actions and provisioning actions.

The main expected disadvantages are: Time taken for convergence; Configuration fine-tuning complexity; Mutation parameter definition; Fitness normalization or selection parameter configured sometimes by trial and error. Also, the algorithm may obtain incomprehensible solutions. The results could be out of scope, inefficient or incomprehensible from an engineering point of view. As a result of these disadvantages, the fitness function should be carefully designed and optimized.

The fitness function measures the differences between individuals and, therefore, decides which one is better. Some important characteristic are explained by Rebecca J. Parsons [209, Chapter 9], indicating that the fitness function is the way to discriminate between individuals and internal states, and it is desirable that similar fitness values are obtained for individuals with similar shared characteristics.

We do not aim to reach a fitness function where parameters are calculated and optimized with an evolutionary algorithm. Our proposal tries to distribute auto-scaling actions among existing containers to prove, in a preparatory work, that distributed auto-scaling represents a potential solution that mimics what happens in some living ecosystems.

```
while True do
    Wait (T);
    p=GetRandomNumber(0,1);
    NOX=GetNOXValue();
    if (NOXbetween(SO_L, 100)) ∧ (p >= SOp) then
        Scale-out Action;
    else if (NOXbetween(SU_L, SO_L)) ∧ (p >= SUp) then
        Scale-up Action;
    else if (NOXbetween(SI_L, SD_L)) ∧ (p >= SDp) then
        Scale-down Action;
    else if (NOXbetween(0, SI_L)) ∧ (p >= SIp) then
        Scale-in Action;
    else
        No Action;
    end
end
```

Algorithm 1: Cell algorithm for every decision in auto-scaling module.

6.4 Proposed models

For testing purposes, an application is a group of loosely coupled, fine-grained stateless services that communicate among them using a lightweight protocol. Requests are distributed into the application by means of a load balancer (Figure 6.1), which receives the user requests. Unless otherwise stated, experimental simulations use a round-robin load balancing algorithm, though this is changed for other distribution mechanisms to analyze the system behavior in other circumstances.

The initial proposed model has five possible environment adaptation actions on a classic scalable web architecture, as shown in Figure 6.2: horizontal scaling - *scale out* (denoted by SO), horizontal scaling - *scale in* (SI), vertical scaling - *scale up* (SU), vertical scaling - *scale down* (SD) and no action (N).

Furthermore, probability functions are used in order to decide whether an action should be applied or not. This converts our system into a stochastic process. After a decision is selected (SO, SI, SU or SD), each cell performs the defined adaptation, for a time t, only if the probability is greater than a random value generated internally at the time for every defined action. Therefore, four probabilities are defined for four

potential actions. SIp is the probability to become dead in horizontal scaling (*scale in*), SDp is the probability of adaptation in the case of vertical scaling (*scale down*), SUp probability for vertical scaling, but in *scale up* and SOp is the probability to perform an action of horizontal scaling (*scale out*), as described in Figure 6.2. Notice that the probability of these actions prevents an action to be concurrently taken by all the containers, as it would happen, for example, during a spiky workload increase. Therefore, it can be interpreted as the ability to adapt the system for every action. See Algorithm 1 for a cell algorithm for auto-scaling decision.

6.4.1 NOrmalized eXtended resource metrics (NOX)

Normalisation is required in order to compare different metric values, which are referenced as X_t in time instant t. This NOX (NOrmalized eXtended resource metric) is created based on the original metric values as described in equation 6.1.

$$X_t = a_0 * f_0(P_{0_t}) + a_1 * f_1(P_{1_t}) + ... + a_n * f_n(P_{n_t}) \quad (6.1)$$

where $a_x >= 0$ indicates the participation of every function (f_n) in NOX, $a_0 + a_1 + ... + a_n = 1$, P_{x_t} represents the parameters for sub-functions in time instant t and function $f_x <= 100$, $x \in [0..n]$ (n represents the total number of components in the equation). It takes values from 0 to 100 allowing us to compare systems in the way we desire. This sub-function represents a part of the NOX function with different treatments according to the variables defined (e.g. $f_1 = \%CPUload$, $f_2 = \%RAMuse$, $f_3 = IO\ operations * 100\ /\ max\ IO\ operations$, etc.).

The desirable features for NOX values are described by Khurana [136] for software metrics, among others: Consistent and objective, easy to calibrate, easy to obtain and robust.

6.4.2 Auto-scaling Self-sufficient Cell Model (SCM)

A first analysis is made trying to maintain cells as independent as possible. The main feature of this model is the lack of direct interaction among the cells. The two main characteristics of the SCM algorithm are the NOX function applied (see section 6.4.1) and the transition function used. This initial model is interesting because if we define a NOX function that uses only local parameters, the model would reduce container data interchange and network traffic.

The SCM model has multiple options relaying on the NOX function, action limits or data from the previous status. Some initial alternatives are explained as follows:

- SCM-A option. The transition function is composed of a NOX value that only considers %CPU used ($X_t = \%CPU$) and a simple transition function (see Table 6.1) where X_t represents the NOX function. In this table every line represents a range of response and possible action depending on the NOX value. The column "Likely cell reaction" represents the name in a bio-inspired scenario whereas the "Auto-scaling name" column is the action name in an auto-scaling scenario. The last column ("Action") is a representation of the activity carried out.

  After the NOX evaluation and the transition function, an action is obtained (SO, SI, N, SU or SD), and the probabilistic correction explained at the beginning of section 6.4 is carried out to prevent that every cell performs the same action. In Table 6.1, the variables SI_L, SD_L, SU_L, SO_L represent the thresholds defined in the model in order to take actions. SI_L is the upper limit of dead range action. Values of X_t between SI_L and SD_L cause *scale down* actions and SU_L is the lower limit for *scale up* whereas SO_L is the upper limit. The upper limit is defined by SO_L that is the lower threshold for *scale out* actions.

Function	Likely cell reaction	Auto-scaling name	Action
$SO_L <= X_t <= 100$	Reproduction	Horizontal scaling (*scale out*)	SO
$SU_L <= X_t < SO_L$	Adaptation	Vertical scaling (*scale up*)	SU
$SD_L <= X_t < SU_L$	Not action	–	N
$SI_L <= X_t < SD_L$	Adaptation	Vertical scaling (*scale down*)	SD
$0 <= X_t < SI_L$	Dead	Horizontal scaling (*scale in*)	SI

Tabla 6.1 Transition function definition for the SCM-A option.

- SCM-B option. In order to simplify the possible actions applied to the managed system, the "Not action" (N) is removed. This case is an adaptation of SCM-A, where the range *Not Action* disappears and vertical scaling actions are joined to create alternative options using the NOX value from the previous iteration (X_{t-1}). The transition function is defined in Table 6.2. Variables SO_L and SI_L maintain the same meaning and vertical actions are made according to the

previous NOX function. Therefore, we are prioritizing horizontal scaling and keeping limited the vertical scaling actions.

Function	Likely cell reaction	Auto-scaling name	Action
$SO_L <= X_t <= 100$	Reproduction	Horizontal scaling (*scale out*)	SO
$(SI_L <= X_t < SO_L)$ $and (X_{t-1} <= X_t)$	Adaptation	Vertical scaling (*scale up*)	SU
$(SI_L <= X_t < SO_L)$ $and (X_{t-1} > X_t)$	Adaptation	Vertical scaling (*scale down*)	SD
$0 <= X_t < SI_L$	Dead	Horizontal scaling (*scale in*)	SI

Tabla 6.2 Transition function definition for the SCM-B option.

- SCM-C option. The transition function is the one defined for SCM-B in Table 6.2 and the NOX function is $X_t = 100 * \frac{Qsize}{Qlimit}$. In scenarios involving a web application where requests are demanded not periodically (with an unpredictable not periodic pattern in time), Q_{size} is the number of pending FLOPS and Q_{limit} is the value of the maximum number of pending requests.

More complex NOX and transition function can be defined depending on the system response that we need, but these cases explain the typical behaviour of an auto-scaling system with well-known system parameters.

For the sake of clarity and simplicity, all the references to SCM will correspond to the SCM-B option.

6.4.3 Auto-scaling Interactive Cell Model (ICM)

Auto-scaling Interactive Cell Model is an example of a system where cells/containers know information about adjacent cells/containers directly. This requires data interchange among cells, directly or through a mediator service.

Action probabilities and transition functions are similar to the ones described in the SCM models. However, the NOX function definition is different from ICM models. We need a function that joins information about nearby cells and local information. The set of cells closer to a given one is denoted as S_t^n.

$$X_t = \alpha * X_t^{local} + \beta * X_t^{neighbour} \tag{6.2}$$

$$X_t^{neighbour} = a_0 * f_0(S_t^0) + a_1 * f_1(S_t^1) + ... + a_n * f_n(S_t^n) \quad (6.3)$$

where $a_x >= 0$ is the participation of every sub-function in the neighbour part of the NOX function, $a_0 + a_1 + ... + a_n = 1$, $\alpha + \beta = 1$, $\alpha >= 0$ and $\beta >= 0$, $f_x <= 100$, $x \in [0..n]$ and S_t^n represents a set of cells defined at time instant t. When $\beta = 0$ this equals to the SCM model. X_t^{local} is equal to the SCM model in equation 6.1. The transition function is the same used for the SCM-B option (Table 6.2) and the probabilities defined in the interval maintain the same interpretation. The neighbour function $X^{neighbour}$ is based on a linear representation of cell space. This is built with only one dimension, the easiest representation for a n-dimensional space to render cells.

6.5 Models evaluation

After explaining our proposal, we run several simulations to evaluate the accuracy of the models. Tests were performed under controlled conditions explained in the following paragraphs.

In this case, we rely on simulation techniques to show the best performance cases and to avoid error configurations that would render the model unusable. Running these models in a production infrastructure (using real workload, actual hardware and real total number of elements) would involve a large-scale number of resources that are not justified for a first problem approach. In a stochastic system, such as this one, we run simulations several times in order to validate output values with multiple combinations of input parameters. This technique requires a lot of computational power to be executed a relevant number of times, thus discouraging the use of a real production infrastructure.

The first feature observed is that the transition function is related to the NOX function. In our case, the NOX function reviews node execution values and the transition function changes the total number of nodes (horizontal scaling) or node execution parameters (vertical scaling). Relationship between NOX function and transition function must be well designed to prevent wrong behaviours. A well-known case is an incorrect horizontal scaling action in cases of network bottlenecks.

All models were run using a self-implemented code accessible in GitHub [109]. This simulator emulates every node as a class instance that adjusts its parameters using the previously defined transition function.

The following sections validate the models through simulations and verify how the system responds to load changes with the proposed algorithms. After that, we define the process variables that will be used to compare the proposed models. Finally, relationships among variables and the results are confirmed or belied using graphical methods.

6.5.1 Global conditions

The general conditions for testing are described in the following paragraphs together with the methods used for data collecting, monitoring and analyzing.

Measuring Elasticity

Generally, auto-scaling is done to optimize the system in terms of cost and/or performance like the response time or throughput. Indeed, in the work by Ai et al. [3] the authors propose an extended list of values that can be used to evaluate scalability proposals. Workloads in a production system are highly unpredictable and do not typically exhibit clear patterns [140]. Furthermore, without regarding prediction systems, all scalability systems have a non-defined time delay between action and response. Therefore, we needed some strategy to adjust a correct evaluation in addition to a correct response.

We have four statistical metrics for evaluation:

1. **MAE** (Mean Absolute Error). It is a measure of difference between two continuous variables.

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|e_t| \tag{6.4}$$

2. **RMSE** (Root Mean Squared Error). Larger errors have a larger effect on this value. Thus, it is very sensitive to outliers.

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}e_t^2} \tag{6.5}$$

3. **EVS** (Explained Variance Score). The best possible value is 1. Lower values are worse.

$$EVS = 1 - \frac{\sum_{t=1}^{n}(e_t - \bar{e})^2}{\sum_{t=1}^{n}(y_t - \bar{y}))^2} \tag{6.6}$$

4. **R2S** (Coefficient of Determination or R-Squared value). The best possible value is 1 and it can be negative (because the model can be worse).

$$R2S = 1 - \frac{\sum_{t=1}^{n} e_t^2}{\sum_{t=1}^{n}(y_t - \bar{y})^2} \tag{6.7}$$

where: $e_t = V_{real,t} - V_{expected,t}$, $y_t = V_{real,t}$ and $\bar{y} = mean(y_{series})$

Alternatively, Dynamic Time Warping (DTW) is a time series alignment algorithm developed originally for speech recognition by Sakoe and Chiba [207]. It aims at aligning two sequences of feature vectors by warping the time axis iteratively until an optimal match between the two sequences are found. According to our purposes, the best match between two sequences is obtained by measuring the DTW distance. It is proposed as an alternative to e_t calculus for MAE (Equation. 6.4).

In order to compare two time series, DTW distance perfectly fits our purpose. The first time series represents the resources needs in a t time in comparison with a second real-time series equal to our available resources. Then, the DTW distance between the two series is a measure of the adaptive capacity for the system designed (available resources) and system desired (resource needs). When it is near to zero, it is a proper value. Therefore, the greater the value the worst it is considered. However, the DTW algorithm only provides modulus ($|x|$) values comparative (positive distances) for best fitting points. An additional problem is that using modulus values no difference exists when resource load values are greater or lower than the reference series. This implies no evaluation of under-provisioned or over-provisioned behavior, like MAE or RMSE do.

When the load of total requests exceeds the system capacity, the served requests must wait until there is processing time available. The DTW algorithm does not distinguish between over-provisioned and under-provisioned scenarios. To model overload, we introduce a store for requests that are partially processed. The size of this store indicates resource needs to finish the jobs in container (e.g., if 10 MFLOPS of CPU resources are pending, the store size will be 10). To prevent under-provisioned system times from not being analyzed in our study, we included a new result value for all simulations called queue size. In this case, the total queue size (Q_{size}) is the value selected for system evaluation. This experiment was designed for testing purposes,

validating that over-provisioned systems have $Q_{size} = 0$ values for all the executions and under-provisioned show $Q_{size} > 0$.

Algorithms for testing and comparation

Together with our bio-inspired algorithm proposal, we used two well-known centralized algorithms too:

- *Common.* This simple algorithm allows adapting capacity depending on $CPU_{utilization}$ like Amazon's Simple and Step Policies EC2 Auto Scaling feature (SS) [120]. It is simple and easy to code, but beneficial and efficient for simple applications. When $CPU_{utilization}$ is under 30%, we reduce the total number of containers running; over 70% we create a new container.

- *Prediction.* This algorithm is similar to the previous one, but we tried to estimate future values using SARIMA(5,1,0), also known as seasonal ARIMA, implemented in Python's *statsmodels* library to forecast the CPU usage in the next time slot instead of using the actual $CPU_{utilization}$. SARIMA prediction uses all series of data available in order to estimate the next value. After this calculus, we use similar limits to those used for the Common algorithm introducing proportionality to container creation. If the load predicted by the algorithm is not satisfied by adding only one node then it will create the necessary additional nodes.

Servers load for measuring

We analyze our proposal using three time series. The first load used is a synthetic proposal -SYNTLoad- (see Figure 6.3), where different web-based workload patterns across time are shown. Based on the work by McNaughton [172] and Gill et al. [94], we take several workload patterns for testing purposes. This creates a non-cyclic and non-seasonal time series that will serve us to better evaluate the models and configurations with unpredictable workloads. The second one is a log from August 04 to 31, 1995 of HTTP requests to the NASA Kennedy Space Center WWW server in Florida -NASA95- (see Figure 6.4) [148]. Finally, we have a log from France FIFA World Cup Series 1998 -FIFA98- (see Figure 6.5) [149]. With the aim of reducing real load extension from FIFA98 and NASA95, the total lines are grouped in a single request for each 10 minutes. Therefore, the number of requests is the same, but they are grouped in longer time periods.

To assess the models, in the simulator, load requests are transformed as resource load using configuration variables. All executions carried out were configured to use

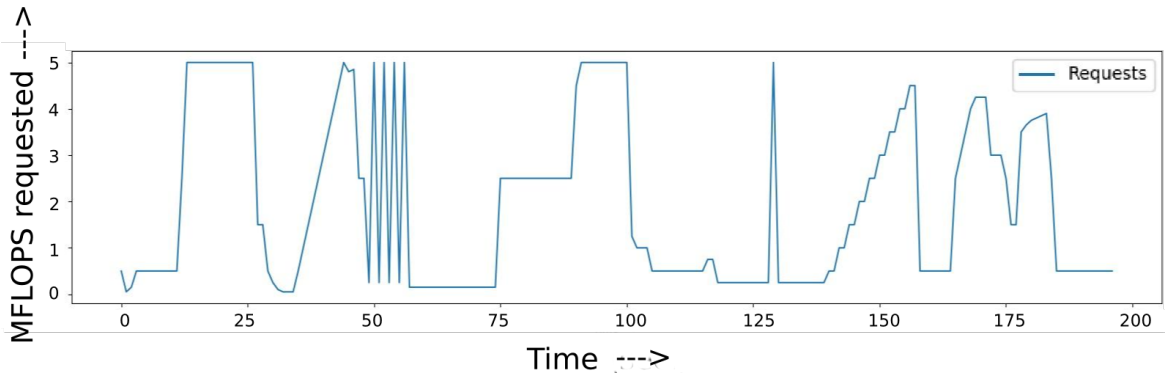


Fig. 6.3 Synthetic load series (SYNTLoad).

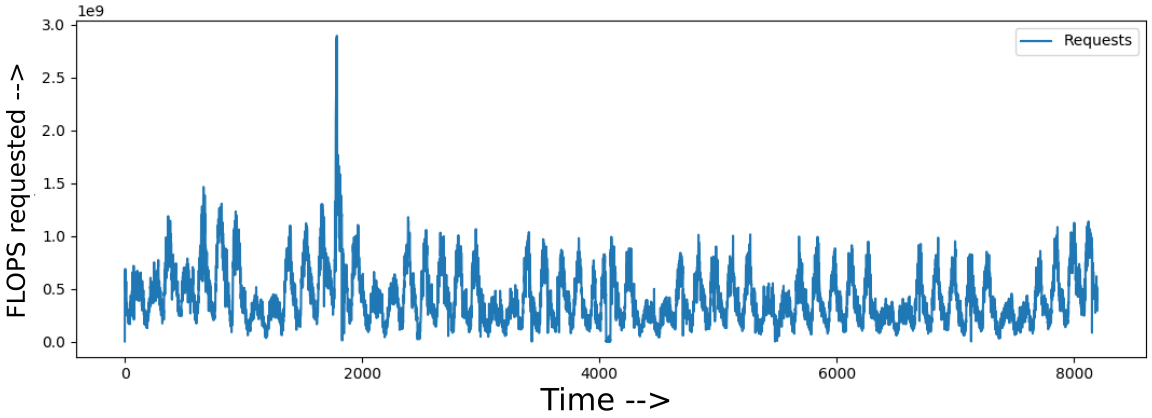


Fig. 6.4 1995 NASA series (NASA95).

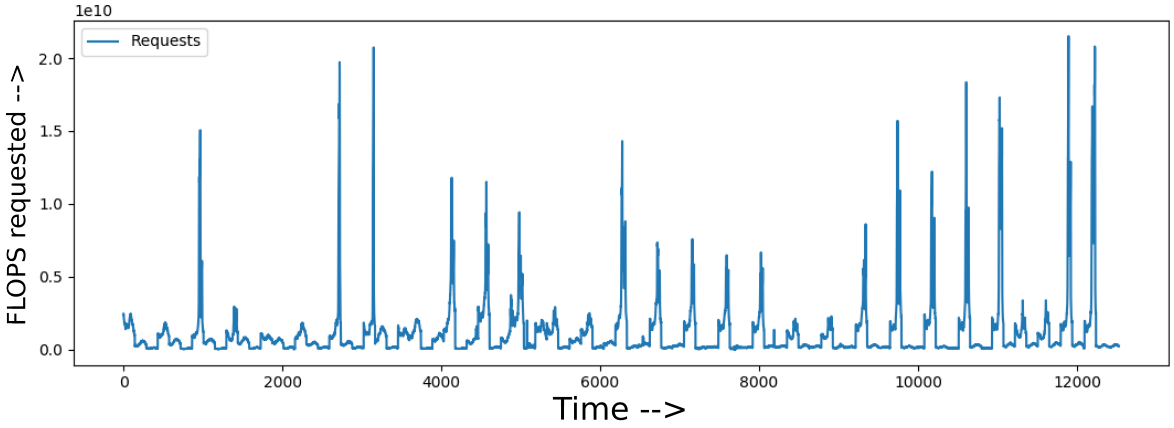


Fig. 6.5 1998 FIFA World Cup series (FIFA98).

FLOPS as the basic measurement unit for output values, summaries and visualization data for the sake of better understanding.

6.5.2 Selecting best proposed algorithm

So far, we have presented models and the way to measure and compare them. Firstly, to validate our proposal, we evaluate models with SYNTLoad in this section. After that, in section 6.5.3, we will compare the best proposed model with real loads (FIFA98 and NASA95) and well-known auto-scaling algorithms (common and prediction).

Previously to experiment validation, we tried to relate parameters provided on simulation and outputs (see Figure 6.6). In this case, varying five input simulation parameters (SI_L, SI_p, SO_L, SO_p and SD_p or SU_p) the range of parameters are $SI_L \in [0..100]$, $SI_p \in [0..1]$, $SO_L \in]SI_L..100]$, $SO_p \in [0..1]$, SD_p or $SU_p \in [0..1]$ (see Table 6.5 compared with Table 6.4). We produced two outputs in a controlled environment, aggregate DTW distance and #containers, both shown as columns. All other possible parameters in the simulation were maintained with constant values. As we use percentage variables (SI_p, SO_p and SD_p or SU_p) to obtain a result, we run ten times the same simulation and the result was the average of all the executions. Rows are identified as limits (SI_L and SO_L) or probabilities (SI_p, SO_p and SD_p or SU_p). Limits use steps of 10 whereas probabilities have a step of 0.1 and both are represented in the X-axis and normalized in ranges from 1 to 100. Y-axis is normalized too. By varying parameter values in the X-axis, we created all the combinations for DTW distance and *#containers* (represented in the Y-axis). The relation between DTW distance and #containers can be obtained as the result of previous variations and is represented in the lower part of the multivariate analysis.

Also, we generate Figure 6.7 with a multivariate analysis of independent interval variables $MinCellsRunning$, $MinVerticalLimit$ and $MaxVerticalLimit$ related to the dependent variables DTW distance difference (measured in GFLOPS) and total number of containers (#containers) produced in the simulation.

Multivariate analysis involves observation of more than one outcome variable at a time in order to reduce a large number of variables to a smaller number of factors. The relation between variables and results (aggregate DTW distance and #containers) are not clear in most cases. Only SI_L, SI_p and $MinCellsRunning$ variables have an influence on the results (dependent variables). It looks a good variable related to Scale-in actions (SI). Only with a low value of SI_L a good auto-scaling response is achieved with an appropriate number of containers. The best value of $MinCellsRunning$ is around 9.

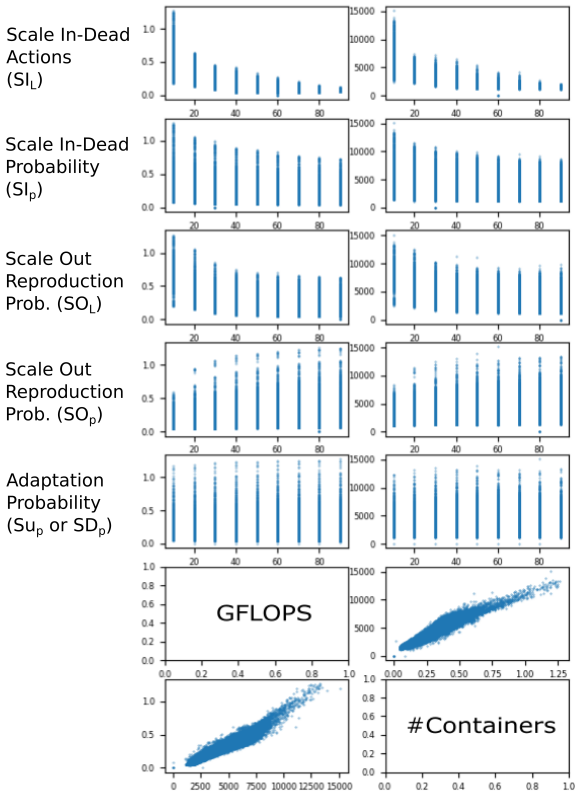


Fig. 6.6 First multivariable analysis for SCM simulation.

Measuring SCM experiments

This section validates the SCM model for auto-scaling systems in order to gain better understanding about its behavior under SYNTLoad situations (Figure 6.3). The dynamic container provisioning output for executions is shown in Figure 6.8. In the same way, for all experiments, we capture the DTW distance (total distance lines between the load applied and the system capacity obtained) as the main approach to compare experiments.

The **SCM1 experiment** runs the SCM-B model with constant variables. Steady values are presented in Table 6.5. The first line shows three actions that can be performed (*SI - scale in / SD - scale down* or *SU - scale up / SO - scale out*). The second line displays function limits (0, SI_L=30, SO_L=90, 100) and the last row includes final action percentage (*SIp* - Dead Probability (70%), *SUp=SDp* (50%), *SOp* - probability of *scale out* (10%)). The limit between the *SD* and *SU* actions is explained in Table 6.2. Reading Table 6.5 by columns: *SI* action (*scale in*) is triggered when NOX value is from 0 to 30 with a probability of 70%. *SU* or *SD* action is triggered when NOX value is between values 30 and 90, with a probability of 50%.

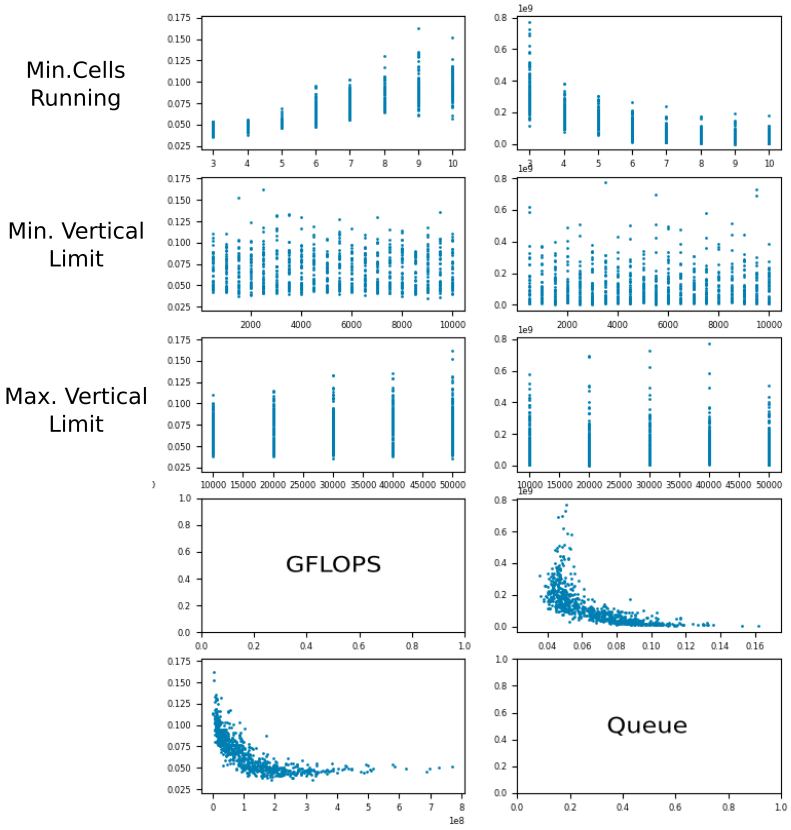


Fig. 6.7 Second multivariable analysis for SCM simulation.

Finally, the *SO* action (*scale out*) is triggered with values of NOX between 90 and 100 and only 10% of the times. These values are selected initially as a starting point for the system in a heuristic way. The following experiments adapt these values. The decision for vertical scaling is made based on previous resource load. If the preceding load is less that the current resource load requested then the vertical decision aims at increasing the capacity (SU). Conversely, the decision is to decrease (SD) the resources allocated to the cell. Figure 6.3 (SYNTLoad) shows the total load demanded for total requests in our experiment. See Table 6.3 for detailed results of SCM1 experiment.

In the **SCM2 experiment**, the load balancing algorithm is modified. Preceding experiments involved simulations using a round-robin load balancing algorithm. In this experiment, an agent based algorithm for load balancing is integrated in the simulator. It distributes requests based on the container load. With this approach, even if we assume the same load in every request, not all container will receive the same number of requests. Instead, we use the processing capacity for all nodes as a heuristic to distribute the requests among the least loaded cells. Using only horizontal scaling, no

Experiment	Tot. DTW distance	Tot. processing containers	Ratio Q_{size} / #containers	Total Q_{size}
SCM1	194	4,986	1.492	7,440
SCM2	57	1,125	0.005	6
SCM3	418	7,493	0.021	158
ICM1	519	11,884	0.072	864

Tabla 6.3 Compilation of featured experiment results.

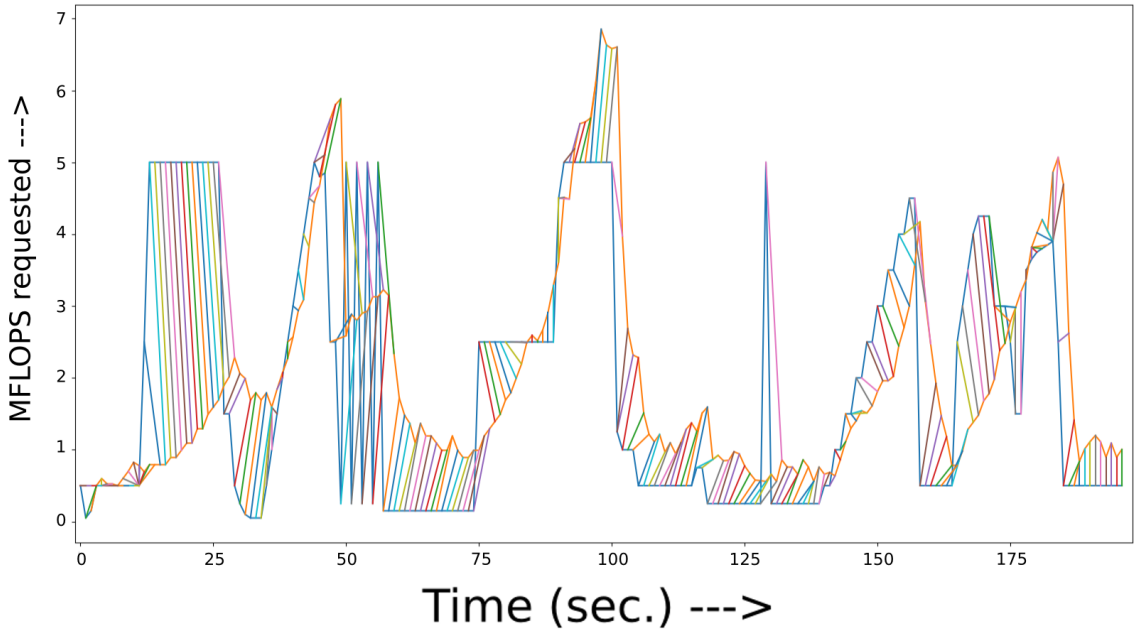


Fig. 6.8 DTW distance results for SCM-B simulation.

Action code	SI		SD or SU		SO	
NOX limits	0 <—->	SI_L	<—>	SO_L	<—>	100
Act.percentage	SI_p		SD_p or SU_p		SO_p	

Tabla 6.4 ICM1 experiment limits and values. Including variables options.

Action code	SI		SD or SU		SO	
NOX limits	0 <—->	30	<—->	90	<—->	100
Act.percentage	70%		50%		10%	

Tabla 6.5 SCM1 experiment limits and values.

capacity differences among nodes exists. All the nodes feature the same processing capacity. To enable nodes capacity variation, we need vertical scaling that does not exist if the dead superior limit variable (SI_L) and horizontal lower limit variable (SO_L) are the same (center range is reduced to zero). See Table 6.3 for detailed results of SCM2 experiment.

SCM3 experiment. For this experiment, we created a system with improved characteristics in vertical scaling. Indeed, scaling up/down (vertical scaling) is sometimes more efficient than scaling in/out (horizontal scaling) (see Al-Dhuraibi et al. [5]). Indeed, adding a new node requires more time than allocating further CPU, memory or other resources to an existing one. Therefore, we design the experiment favouring additional vertical scaling actions compared to the previous experiment. The parameters used in this experiment will reduce the range for SI and SO as much as possible, setting SI_L to 5 and SO_L to 95. This creates a wider range for vertical scaling (from 95 to 5). The request time series is the same one used in all the previous experiments: SYNTLoad.

Measuring ICM experiments

This section introduces the ICM model under basic setup conditions.

ICM1 experiment tries to improve the results obtained in the SCM experiments and apply the ICM model explained in section 6.4.3 with new external data sources. ICM cells collaborate directly with other cells to improve total throughput and reduce total Q_{size}. In this case, we use a straight line (one dimension) living space where cells remain until death. Every cell has only cells to the left side and to the right side because the space is continuously configured where the last cell has the first cell as neighbour and vice versa. When a cell looks for cells adjacently it is possible to see a distance greater than 1. The parameter *icm_size* indicates the total cells seen on both sides used for the computations. Other options, such as a two-dimensional space or a multidimensional mesh are possible, but they are not addressed in this document because the resulting analysis is more complex.This experiment brings in a major problem when we consider intercommunication between neighbour containers. The use of a simulation environment allows us to evaluate models independently from the communication strategy among the cells, which can be achieved by a message bus or a service mesh.

We focus on whether horizontal scaling is enough to adjust to workloads and compare the results against other experiments. Therefore, in this model, no vertical scaling (up or down) is produced, all adaptation is made using horizontal scaling

Load Name	FIFA98		NASA95	
Vertical Auto-scaling	With	Without	With	Without
Total Distance DTW	11,165,275	10,782,642	15,425,883	2,759,028
Total Q_{size}	4,603	16,505	1,598,359	2,188
Total Container Used	446,390	2,203,148	3,357,354	576,863

Tabla 6.6 Summary table comparing bio-algorithms with vertical scaling (SCM2) and without vertical scaling (SCM3).

(using in/out scaling), SI or SO decisions. The NOX formula is similar to the one used for SCM-B but now the value is a combination of local data and neighbour's information. All information about local resources used can be mixed with all average information from then right side and then left side cells using a weights formula (see equation 6.2 parameters X_t^{local} and $X_t^{neighbour}$). Value for %Local is $\alpha = 15$, this means a %neighborhood value for $\beta = 85$.

For testing purposes, all the information from neighbours is obtained from a centralised database periodically populated with the information from the neighbours themselves. See Table 6.3 for detailed results of the ICM1 experiment.

Results analysis

The results in Table 6.3 highlight important details comparing the four data experiments. According to the Tot. DTW distance column, the best algorithm is SCM2, where the result obtained is the closest to 0. The experiment that uses the lowest number of containers to run the workload is SCM2 as well, using 1,125 containers across all the experiment. Considering the Q_{size} column, the lowest size is achieved in experiment SCM2. Therefore, SCM2 stands out as the best approach in the model proposed. ICM1 uses a large number of containers (11,884), a 58% more than SCM3 (7,493). SCM1 was significantly worse at Q_{size}. The ratio $Q_{size}/\#containers$, highlights a similar situation: the best ratio is obtained for SCM2, whereas SCM3 and ICM1 show similar response times and the worst is SCM1. Therefore, mixing a bio-inspired approach with a local balancing strategy stands out as the best approach.

6.5.3 Comparing with real world loads

This subsection describes the results obtained as a combination among loads (explained in previous section 6.5.1) and algorithms (two exposed in section 6.5.1 and models

proposal of section 6.4.2). Trying to explain the most important data in Table 6.7 by columns.

SYNTLoad: Notice that, in Table 6.7, the Total Container Used in Bio (running SCM2 option) is higher than the one used in other real world algorithms used (Common or Prediction) but Total Q_{size} is lower. This offers an explanation of why a lower Q_{size} is obtained for this proposal. Using DTW, instead of evaluating $error = V_{expected} - V_{actual}$ to calculate MAE, we get similar values for three proposals.

FIFA98 Load: Bio algorithm was run using a combination of Vertical and Horizontal Scaling (see 6.5.3). The values shown for other variables are similar to the Bio algorithm. MAE DTW, MAE and RMSE are similar too, but EVS and R2S not. The value of Q_{size}, related to response time, is better for our Bio algorithm. Predictive worst results are justified for a poor forecast and not performing optimal seasonal adjustment of time series analysis with outliers. Indeed, *Common* is a very simple algorithm to adjust to a complex workload.

NASA95Load: Bio algorithm was run using only Horizontal Scaling (see 6.5.3). This load shows differences with respect to other two loads. It is true that Bio uses twice as many containers, but Q_{size} is very low compared with other algorithms. MAE DTW has the worst values for Bio too. The Predictive algorithm is better in this case, but Common also obtains a good result.

Confirm Horizontal or Vertical Scaling best results in real world

In previous sections and Table 6.3, we compared models using a SYNTLoad. To confirm the results, we run the best modes (SCM2 and SCM3) with real loads. This generates Table 6.6 where the most important values are presented. For FIFA98, the best results correspond to vertical-horizontal mix scaling option (SCM2) but, for NASA95, the best results are without vertical scaling (only horizontal) using SCM3. We estimate that this is due to the load peaks in FIFA98 that are not present in NASA95. Our best performance hypothesis for Vertical Scaling, tested on SCM3, is not confirmed and performance is clearly subordinate to load type.

Load/Algorithm combination Outcomes

The best results for our proposed model are accomplished in loads with unpredictable series points called outliers. Most predictable loads are better for other two algorithms. DTW used, as error measure, is useful to compare the shapes of series in unpredictable cases. Our Bio algorithms are always over-provisioned, and they have better measures for response times that need be validated in further research.

Load Name	SYNTLoad			FIFA98			NASA95		
Auto-scaling algorithm	Common	Bio	Predic.	Common	Bio [c]	Predic.	Common	Bio	Predic.
Auto-scaling # points		197			12,528			8,199	
Total Distance DTW	5,710	7,006	5,778	11,144,499	11,165,275	11,121,795	1,380,128	2,759,028	1,412,102
Total Container Used	5,767	17,596	5,780	1,104,654	466,390	1,108,445	295,180	576,863	291,925
Total Q_{size}	196	5	193	3,095,636	4,603	2,956,310	750,523	2,188	823,881
MAE DTW [a]	29.0	35.6	29.3	889.6	891,2	887.8	168.3	336.5	172.2
MAE [a]	31.4	42,2	31,6	908.5	911.5	906.5	174.1	381.6	177.2
RMSE [a]	37,2	61,4	37,2	1,792.6	2,001.1	1,789.9	228.9	588.6	232.6
EVS	0.048	-0.872	0.029	0.140	0.631	0.142	0.282	-2.952	0.254
R2S	-0.039	-1.825	-0.035	0.129	-0.084	0.132	0.225	-4.123	0.199
Capacity Coef.[b]	2.44	0.35	2.33	6,83	0.66	6.81	3.40	0.32	3,45

[a] x1.000.000

[b] Coefficient for Maximum Requests/Maximun Processing Capacity

[c] Using Vertical and Horizontal Auto-scaling

Tabla 6.7 Summary table comparing loads and auto-scaling algorithms results.

By putting response time first, using a Bio algorithm is the best option. Contrasting loads, if it is foreseeable, a Prediction algorithm or a Common algorithm are better selections, but if we have a non-cyclic and non-seasonal time series load, Bio options would be a possible selection as a result of a better system performance adjustment using an over-dimensioned number of containers but it clearly increases the cost of the solution.

Experiment	SYNTLoad Bio	FIFA98 Bio	NASA95 Bio
Auto-scaling points	197	12,528	8,199
Total container used	17,596	466,390	576,863
Mean containers	89.31	37.22	70.35
Container peak	351	2,911	910
Max # scale-out/in actions	92/280	1,354/1,379	440/748
Mean container lifetime	48	1,566	132

Tabla 6.8 Scale cost for bio-algorithm.

Feasibility of Scale actions

If we want an agile response to change, we need to estimate the workload according to previous data or design a fast response time system (with Q_{size} low). As a final assessment of our Bio proposal, we need to analyze the economics of scale. On Table 6.8, we reference not only the containers used, from Table 6.7, but also variations and peaks in the simulation.

Adding, removing or modifying container characteristics is a task that we need to measure when using a high number of containers in our proposal. We need an appraisal of the cost of running containers taking into account that the Bio algorithm increases the total adaptation actions made as well.

Vertical scaling is an elementary response action for a temporary lack of resources, adding more CPU capacity or memory to our under-provisioned node. Being a very limited action in time, this has not an effective cost for computing platforms, but vertical scalability has a limit and the cost gradually increases (see Henderson [104], OpenShift and [185]). In container platforms with pay-per-use model some vertical resizing is made using a stop-start model with no possibility of live vertical scaling. For example, Amazon does not support live vertical scaling of EC2 instances. In contrast, we have Jelastic [128] that define a vertical scaling like our proposal based in a scaling limit of resources in two parts, reserved and dynamic.

The cost of horizontal scaling includes the total number of containers running in a time period and our solution sometimes doubles the number of containers with respect to other strategies. But bio-inspired reduces the total time that a container is up. This means that it creates a large number of short-lived containers. Using a Bio algorithm with loads based on peaks, we ensure right scaling and controlled cost.

Another important issue is container creation times. In [101], authors create 1 million containers in 266.7 s. This is a rate of nearly 3,750 containers per second. Greater container scale-out demand is 1,354 containers for a slot time (similar for scale-in). Focusin on Container Orchestration Platforms there was no noticeable difference in the launch time using 30,000 containers in Docker Swarm or Kubernetes (see [164] [184] [86]).

Sharper load peaks or series with greater number of outliers are solved better with horizontal scaling and repetitive load are conveniently solved with vertical scaling. To put the two possibilities together, a possible solution is to equip our models with a peak detection in request series.

6.6 Conclusions

An early study was made to handle a bio-inspired distributed algorithm in a container orchestration platform by employing a simulation software, introducing two experimental models that help to distribute decision making across the involved nodes.

Achieving low Q_{size} values, whereas exhibiting a high processing capacity (low DTW distance) are opposite goals. In some cases, attempting to improve a metric by configuring certain parameters results in worsening other metrics. A compromise solution is the most appropriate solution. Oscillation problems of auto-scaling algorithms are limited in bio-inspired models and appear to be controlled by configuration parameters. Suitable models mix vertical and horizontal scaling, joining all actions in an easy way and allowing other potential scaling actions. NOX functions are defined as a tool to engage reactive algorithms or self-sufficient with cell-related models. An algorithm, designed initially for time series alignment in speech recognition (DTW), is used to compare multiple options. We perceive it as a good alternative to equate complex time series produced in auto-scaling actions (very long sets or very detailed sequences).

Bio-inspired models do not offer an upper bound for the number of containers running at a t time. Scale up operations (SU) can be done indefinitely until the total capacity of the hardware is exceeded. Furthermore, scale out (SO) operations

can be unlimitedly performed for all cells. Dead prevention for an inferior limit of total containers (to avoid selection of SI action for all cells) is complex too. The SCM model reveals features of a very sensitive system to multiple parameters. A proper configuration is a multiple parameter combination where it is not easy to calibrate dynamic response to multiple situations. The correct configuration of a multiple parameter combination can lead us to a proper response to unpredictable loads. However, adaptive capacity is one of the fundamental appended values of a bio-inspired auto-scaling system. A matching among real scaling algorithms and bio-inspired, using real-world load series, allowed us to identify the suitability for specific workloads. In two cases Bio-inspired are better: in response to an not predictive and no temporal load or when the requests produce outliers (sudden load increases).

The model introduced exhibits an over-provisioned behavior, with a trend to exceed the requirements, a situation which usually implies a higher cost. There is no cost for creating/stopping actions for software containers in some container services such as Amazon Elastic Container Service (ECS). Thus, Bio model is not penalized in this area since only running resources involve a cost. Indeed, a higher number of nodes for an extended period of time is costly. Our proposal limits peak time and at the same time reduces application response time. More evidences are required to confirm a cost increase, if it is produced.

Finally, additional research lines arise as a result of this work. The first goal is to enhance NOX functions, introducing predictive analysis, pricing models or green computation options. The second line would be focused on understanding and design optimized ways for cells/containers communications and mutual discovery, upgrading the ICM model.

Acknowledgment

The authors would like to thank the Spanish “Ministerio de Economía, Industria y Competitividad” for the project “BigCLOE” with reference number TIN2016-79951-R.

Capítulo 7

Discusión y Resultados

En el presente capítulo se revisan las actividades de transmisión de conocimiento que se han realizado en el ámbito de los estudios de doctorado, así como los productos obtenidos y las colaboraciones realizadas.

7.1 Software desarrollado

- GFAL Cache. Como parte de la investigación realizada en el área de computación Grid, se desarrolló un sistema de cache complementario a GFAL que permitía incorporar las mejoras de una forma sencilla ya que replicaba la API de la librería estándar. Facilitaba incluir en cualquier sistema que ejecutaba en el Grid una caché capaz de explotar la localidad de referencia para el acceso a datos remotos.

- Hadoop Event Detection. Versión operativa compuesta de una concatenación de procesos MapReduce orquestados mediante Apache Zookeeper [85]. Como parte del desarrollo se realizaron los tratamientos necesarios, diseñados mediante MapReduce, para poder evaluar las hipótesis que se plantearon.

- Storm Media Processing. Se desarrolló en Java una completa topología ejecutable en Apache Storm junto con los scripts y procesos de tratamiento de logs necesarios para realizar la interpretación de los datos mediante gráficos.

- COBEATS [109]. Sistema de autoescalado bio-inspirado para Docker. Como parte de la última investigación se ha publicado en un repositorio de libre acceso en GitHub todo el código necesario para su ejecución dentro de contenedores Docker independientes junto con las instrucciones para su parametrización y

ejecución. Se han incluido también todas las fuentes de datos utilizadas para la realización de las pruebas.

7.2 Publicaciones y contribuciones

7.2.1 Conferencias

- 1ª EID (Escuela Internacional de Doctorado) de la Universidad Rey Juan Carlos [44]. En 2018 se presentó una comunicación oral de los primeros avances en el trabajo de elasticidad de los entornos de contenedores. Durante los días 27 y 28 de noviembre de 2018 se participó en la Escuela Internacional de Doctorado que realizó la URJC realizando la comunicación *"Algoritmos de autoescalado bio-inspirados en plataformas de contenedores"*.

- PDP'16 - Heraklion - Creta. Entre los días 17 y 19 de febrero de 2016 se asistió al *24th Euromicro International Conference on Parallel, Distributed and Network-Based Processing* para defender el trabajo realizado en la introducción de multimedia en los sistemas de Big Data. Se defendió el comunicado con nombre *"Detecting Events in Streaming Multimedia with Big Data Techniques"* que luego se publicó en las actas. Se exponen más detalles del encuentro en el Apéndice B.

7.2.2 Artículos publicados

- *"Studying the improving of data locality on distributed Grid applications in bioinformatics"* [105] disponible en el Capítulo 3. Publicado en las actas de Ibergrid'13. Se incluye información adicional sobre los lugares de publicación en el Apéndice B.

- *"Detecting Events in Streaming Multimedia with Big Data Techniques"* [106] disponible en el Capítulo 4. Publicado en las actas de PDP'16.

- *"Logotype Detection in Streaming Multimedia Using Apache Storm"* [108] disponible en el Capítulo 5. Publicado en las actas de PDPTA'17.

- *"Towards Bio-inspired Auto-Scaling Algorithms: An approach for Container Orchestration Platforms"* [107] disponible en el Capítulo 6. Publicado en 2020 por la revista IEEE Access en el tema *Artificial Intelligence in Parallel and Distributed Computing*.

7.3 Colaboraciones

- Internship Tokyo. Entre los meses de septiembre y diciembre de 2015 se realizó una estancia en el *National Institut of Informatic* dentro del LDEAR (*Laboratory of Data Engineering and Archiving*) dirigido por Atshuhiro Takasu para estudiar la incorporación de algoritmos de Big Data en los sistemas de *streaming*. Las líneas principales de investigación en este laboratorio son: ingeniería de datos, *Machine Learning*, *Data Mining*, *Text Mining*, integración de datos e información, recuperación de información y librerías digitales. Como resultado de esta colaboración se realizó la publicación en el congreso PDPTA'17 [108].

- Colaboración con el proyecto BigCLOE durante la realización de todos los estudios. Este proyecto incluye entre algunos de sus fines el desarrollo de infraestructuras de procesamiento de datos masivos y de computación de altas prestaciones. Uno de los ejes de investigación se centra en el diseño e implementación de modelos de elasticidad para frameworks de Big Data, tanto para entornos Cloud como para contenedores software, que permitió la realización de varias publicaciones [106][107][108].

Capítulo 8

Conclusiones

8.1 Conclusiones

Las infraestructuras para el tratamiento de grandes volúmenes de datos han recibido actualizaciones significativas durante los últimos años. Algunas importantes mejoras las podemos encontrar en las arquitecturas subyacentes de los sistemas empleados. Este cambio ha supuesto la adecuación de los algoritmos a las características propias de cada una de ellas. El período de investigación de la tesis se ha orientado al desarrollo de mejores sistemas para el tratamiento de datos, especialmente en su vertiente de mejora de infraestructura pero, en ocasiones, esto ha supuesto la utilización y mejora de algoritmos alternativos.

En un principio se realizó una aportación alrededor de la arquitectura de los sistemas Grid. La investigación se fijó como objetivo la mejora de la ejecución pero el ámbito de trabajo se encontraba entre la mejora del sistema subyacente y la mejora del algoritmo utilizado. Basándose en la susceptibilidad de mejora de las aplicaciones que requieren accesos a pequeñas partes de ficheros de gran tamaño, se obtuvo información relevante cuando se analizó el patrón de acceso a datos de la ejecución del algoritmo de alineamiento de ADN que utiliza la transforma de Burrows-Wheeler. Por ello se implementó y evaluó la mejora del sistema de acceso a ficheros estándar del Grid empleado, agregando una nueva capa arquitectónica que contenía la caché de acceso a ficheros remotos. Esta nueva aportación redujo el tiempo total de acceso y los datos transferidos. Se bosquejó de esta forma una vía para mejorar la infraestructura mediante la librería GFAL y las características que debían tener los procesos para que fueran más eficientes.

La evolución tecnológica llevó a utilizar la arquitectura Lambda de tratamiento de grandes volúmenes de datos. En un primer trabajo se orientó al tratamiento de datos

multimedia mediante estructuras de Big Data cuando éstas estaban tradicionalmente orientadas a los datos tabulares. De esta forma se realizó una prueba de concepto y un flujo de trabajo que extrajo información multimedia de estaciones de audio en forma de *streaming*. Este proceso distribuido, basado en Hadoop, se ejecutó con el fin de mostrar datos de análisis gráfico. El extenso experimento demostró que el flujo de trabajo se puede procesar de forma paralela con la infraestructura propuesta, reduciendo el tiempo de trabajo en un Cloud *on-premises*. De esta forma se demostró su posible funcionamiento, se analizaron los problemas que aparecen al utilizar la infraestructura para la cual no está diseñada y se expusieron posibles soluciones y alternativas.

En el tercer trabajo, esta vez con la arquitectura Kappa, es donde se dio un paso más en la evolución de los sistemas de tratamiento. En este caso, el núcleo de trabajo estaba en un sistema de *streaming* (Apache Storm). Con este tratamiento se buscó analizar la capacidad de esta infraestructura para lograr el tratamiento multimedia encontrando sistemas válidos para grandes volúmenes de datos. Se realizaron múltiples experimentos para mostrar que el uso del *Bloom Filter* reduce la necesidad de uso de memoria en el proceso de cálculo de los resultados. También se demostró que la transferencia directa de imágenes entre *bolts* era la mejor alternativa para el manejo de volúmenes elevados de imágenes. Uno de los conocimientos más destacados que este tratamiento nos proporcionó, es que resulta muy costoso en recursos computacionales y que el dimensionamiento de la infraestructura debe ser acorde al algoritmo de reconocimiento que deseemos utilizar, que es independiente en todo momento de la arquitectura. Se pudo concluir que es posible usar una infraestructura que utiliza Apache Storm para convertir el análisis de vídeo en tiempo real en un problema de tratamiento de grandes volúmenes de información.

Con estos dos últimos tratamientos en las arquitecturas Lambda y Kappa se logró acercar el mundo de los datos multimedia a las arquitecturas generalmente orientadas a los datos textuales o tabulares.

Se finalizaron las aportaciones con la realización de un estudio de los sistemas menos analizados. Se trata de contribuir en el análisis de grandes volúmenes en el área de los microservicios. Una de las posibles mejoras a realizar en los procesos se centró en la escalabilidad de los sistemas de procesamiento. En la cuarta aportación se introdujo un sistema de autoescalado basado en un algoritmo bio-inspirado. Aunque el autoescalado ya se usa en las máquinas virtuales de forma centralizada, el sistema de autoescalado distribuido se definió en el ámbito de contenedores software.

Se analizó el sistema de autoescalado centrándose en un algoritmo distribuido que no tuviera dependencias de componentes centralizados. Se incluyó una aportación nove-

dosa basada en sistemas autónomos que escalaban individualmente, de forma autónoma o relacionándose con otros contenedores. Se realizaron comparaciones entre los distintos métodos y se identificaron los mejores parámetros para su configuración. Con posterioridad se proporcionó una amplia comparativa con los sistemas de autoescalado centralizados existentes en la actualidad.

Se observó una ventaja en aquellos casos en los que las cargas de trabajo tienen un comportamiento no predecible, lo que es de provecho en casos en los que se desconoce el comportamiento o la carga que va a tener en el futuro. Se incorporó, en el mismo estudio, un sistema de medición de las diferencias en la escalabilidad basado en el algoritmo DTW que permitió normalizar una medición más exacta y automatizada de las diferencias en el comportamiento de las opciones en sistemas de escalado.

Una vía para escalar sistemas heterogéneos en plataformas de gestión de contenedores es la utilización de la autogestión. De esta forma los diferentes sistemas son capaces de determinar de forma autónoma, y más eficiente, las necesidades de los algoritmos más adecuados en cada instante de la ejecución y no únicamente en base a los valores de la infraestructura subyacente. Esto amplía el campo de aplicación a numerosos sistemas, arquitecturas y algoritmos, y no se centran tan solo en la mejora exclusiva de un procedimiento.

8.2 Aplicabilidad de los estudios

Se debe analizar la idoneidad de los estudios desarrollados y de los resultados obtenidos como una parte fundamental de cualquier período de investigación. Es por ello que en esta sección analizaremos la posibilidad de usar los conocimientos alcanzados dentro de proyectos productivos y sus posibles aplicaciones en los sistemas reales.

Una de las formas de cuantificar las aportaciones es analizar los trabajos existentes en líneas similares de investigación, aunque no realicen una mención directa al trabajo realizado, pero que tengan alguna conexión con la materia tratada. Es este caso se han extraído las tendencias (en Google Trends) de tres términos que resumen especialmente los estudios realizados (Grid computing, Hadoop y Docker).

Podemos conocer la importancia que tienen los ámbitos de investigación analizando las apariciones de los términos que tenemos en el propio buscador de Google Académico. Para *Grid computing* encontramos 3.340.000 apariciones de documentos, para *Hadoop* obtenemos 198.000, *Docker* son 78.600 menciones, pero para *software container* tenemos 1.660.000 y para el término Big Data se obtienen 5.300.000 menciones. Como vemos

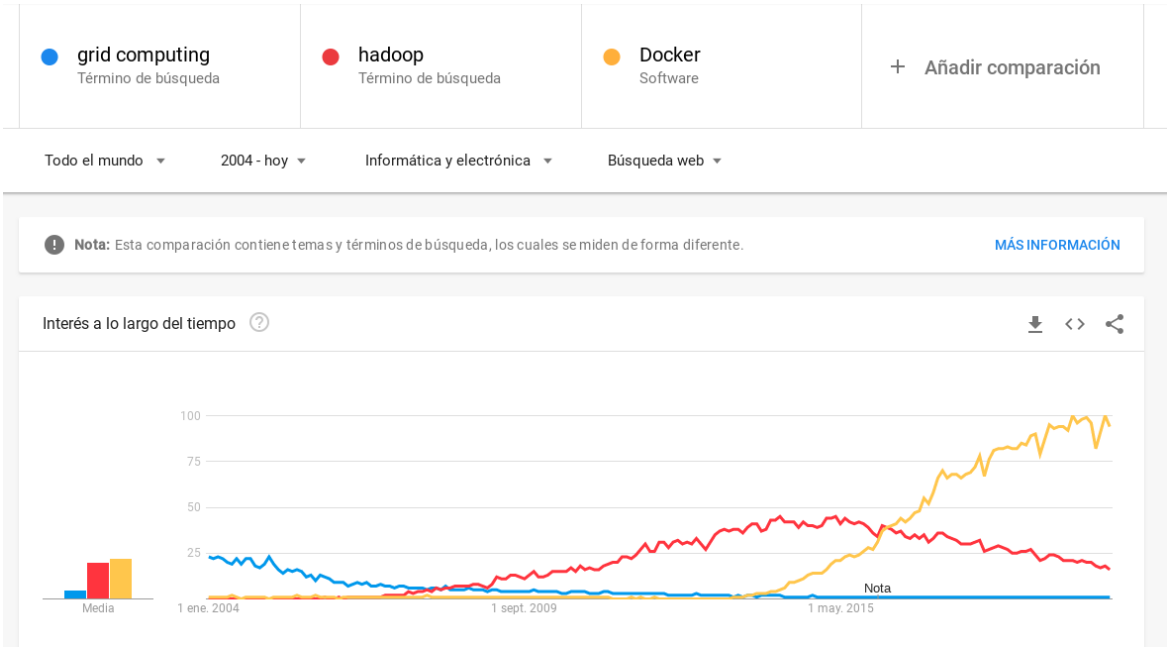


Fig. 8.1 Tendencias en las menciones de términos empleados desde 2004, en todos los países y con referencias en el área de informática (Google Trends a 1/5/2020).

es un ámbito de interés en los trabajos de otros investigadores y así se plasma en la producción de documentos que se han indexado.

Otra forma de analizar el impacto de las acciones de difusión es conocer las citaciones de los artículos publicados. En el caso de los documentos incluidos en esta tesis podemos mencionar *"Detecting Events in Streaming Multimedia with Big Data Techniques"* [106], referenciado en 5 ocasiones.

Una última forma de conocer el interés de estos proyectos es medir aquellos trabajos productivos relacionados y desarrollados posteriormente o bien teniendo en cuenta los proyectos en los que ha tenido repercusión con posterioridad a la publicación del trabajo. Podemos ver trabajos con menciones posteriores para el tratamiento de vídeos con Hadoop [200] [187] [173] [49] [231] [4] [219] [55], con procesamiento de frameworks de streaming [242] [239] [13] y para el tratamiento general de datos multimedia [226]. Referentes a la elasticidad de los contenedores es difícil encontrar posteriores pero es un tema relevante en el último año [204] [100].

Después de estudiar la posibilidad de solicitar una patente como software de control de un sistema rack donde los contenedores pueden escalar libremente y de forma autónoma, esta opción se ha descartado y se ha optado por su exposición pública como software libre bajo licencia Apache [72].

8.3 Perspectivas

Como se ha podido mostrar en el presente documento, se ha producido una evolución de las tecnologías que se utilizan en el procesamiento de grandes volúmenes de datos. Es pues necesario tener en mente siempre esta evolución para poder seguir trabajando en proyectos que resulten novedosos y atractivos en la comunidad científica.

En estos momentos, el desarrollo de plataformas de contenedores es un área en expansión así que son multitud los proyectos que tienen en cuenta la utilización de esta nueva tecnología. Por otro lado debemos tener en cuenta que la utilización de grandes volúmenes de información nos proporciona materia prima para proyectos que pueden ver la luz en un futuro no muy lejano. La convicción de que los algoritmos para grandes cantidades de datos se desarrollarán en los próximos años también harán necesarios nuevos sistemas adaptables de una forma más acorde con las exigencias de procesamiento.

Una de las posibilidades que se abren es la de llevar los trabajos realizados a un entorno productivo. Incluir los desarrollos en una versión lista para un sistema en producción no es un objetivo pero sí que podemos tener una versión para su estudio y configuración.

Otra posible vía de investigación es utilizar métodos de escalado novedosos inspirados en ciencias biológicas. En este caso hay numerosas opciones para autómatas celulares que habría que valorar previamente. De la misma forma hay alternativas que podrían aplicarse a este tipo de problemas como la computación en membrana [188]. La computación de contenedores software abre la puerta a propuestas basadas en problemas en los que se pueden necesitar elementos de cálculo independientes y ejecutables dentro de las infraestructuras de CaaS (*Container as a Service*) públicas.

8.4 Fondos o recursos utilizados en el proceso de investigación

La presente tesis se presenta como recopilación de las actividades de investigación realizadas con las aportaciones de los proyectos y entidades que a continuación se mencionan:

- *National Institute of Informatics* (NII) Japan en su International Internship Program.

- *Council for Science, Technology and Innovation* (CSTI) del gobierno japonés por medio del Cross-ministerial Strategic Innovation Promotion Program (SIP), Infrastructure Maintenance, Renovation, and Management. (funding agency: NEDO).

- Ministerio de Economía, Industria y Competitividad mediante el proyecto BigCLOE con número de referencia TIN2016-79951-R.

- Ministerio de Economía y Competitividad mediante el proyecto CLUVIEM con número de referencia TIN2013-44390-R.

- Infraestructura de cómputo Grid de IberGrid por la cesión de horas de computo y soporte para el uso de los recursos por medio de la *Virtual Organization* (VO) Biomed creada en el contexto de ES-NGI para dar soporte a las actividades de investigación de la comunidad de usuarios de ciencias de la salud y medicina.

- Recursos de procesamiento del Grupo de Grid y Computación de Altas Prestaciones (GRyCAP) del Instituto de Instrumentación para la Imagen Molecular (I3M) centro mixto de investigación de la Universitat Politècnica de València (UPV) y el Consejo Superior de Investigaciones Científicas (CSIC).

Referencias

[1] Martin L Abbott and Michael T Fisher. *The art of scalability: Scalable web architecture, processes, and organizations for the modern enterprise*. Pearson Education, 2009.

[2] Jong Hoon Ahnn. Toward personalized and scalable voice-enabled services powered by big data. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 748–753. IEEE, 2014.

[3] Wei Ai, Kenli Li, Shenglin Lan, Fan Zhang, Jing Mei, Keqin Li, and Rajkumar Buyya. On elasticity measurement in cloud computing. *Scientific Programming*, 2016, 2016.

[4] Nishat Akhtar, Junita Mohamad Saleh, and Clemens Grelck. Parallel processing of image segmentation data using hadoop. *International Journal of Integrated Engineering*, 10(1), 2018.

[5] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. Autonomic vertical elasticity of docker containers with elasticdocker. In *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*, pages 472–479. IEEE, 2017.

[6] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. Elasticity in cloud computing: state of the art and research challenges. *IEEE Transactions on Services Computing*, 11(2):430–447, 2018.

[7] S.T. Allen, P. Pathirana, and M. Jankowski. *Storm Applied: Strategies for Real-time Event Processing*. Manning Publications Company, 2015. ISBN 9781617291890. URL https://books.google.co.jp/books?id=r6K-oAEACAAJ.

[8] Globus Alliance. Our History. https://www.globus.org/our-story, 1997. [Online; accessed 17-05-2020].

[9] Globus Alliance. Globus Toolkit. http://toolkit.globus.org, 1997. [Online; accessed 17-05-2020].

[10] Fernando Alvarruiz, Carlos de Alfonso, Miguel Caballer, and Vicente Hernández. An energy manager for high performance computer clusters. In *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pages 231–238. IEEE, 2012.

[11] Bo An, Victor Lesser, David Irwin, and Michael Zink. Automated negotiation with decommitment for dynamic resource allocation in cloud computing. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 981–988. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[12] Preeti Arora, Shipra Varshney, et al. Analysis of k-means and k-medoids algorithm for big data. *Procedia Computer Science*, 78:507–512, 2016.

[13] Ilaria Bartolini and Marco Patella. A general framework for real-time analysis of massive multimedia streams. *Multimedia Systems*, 24(4):391–406, 2018.

[14] Fabrice Bellard. FFmpeg multimedia framework. https://www.ffmpeg.org/, 2015. [Online; accessed 17-05-2020].

[15] S Binitha, S Siva Sathya, et al. A survey of bio inspired optimization algorithms. *International Journal of Soft Computing and Engineering*, 2(2):137–151, 2012.

[16] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[17] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple object access protocol (SOAP) 1.1. https://www.w3.org/TR/2000/NOTE-SOAP-20000508/, 2000. [Online; accessed 17-05-2020].

[18] John Brooke, Donal Fellows, Kevin Garwood, and Carole Goble. Semantic matching of grid resource descriptions. In *European Across Grids Conference*, pages 240–249. Springer, 2004.

[19] Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm. *SRC Research Report*, 1994.

[20] John Byl. Self-reproduction in small cellular automata. *Physica D: Nonlinear Phenomena*, 34(1-2):295–299, 1989.

[21] Miguel Caballer, Carlos De Alfonso, Fernando Alvarruiz, and GermáN Moltó. Ec3: Elastic cloud computing cluster. *Journal of Computer and System Sciences*, 79(8):1341–1351, 2013.

[22] Nicolo M Calcavecchia, Bogdan A Caprarescu, Elisabetta Di Nitto, Daniel J Dubois, and Dana Petcu. Depas: a decentralized probabilistic algorithm for auto-scaling. *Computing*, 94(8-10):701–730, 2012.

[23] Carlos González Cantalapiedra and Everlyn Vergara Soler. Evolución de los entornos big data y los retos para el arquitecto de datos. *Economía industrial*, 1 (405):21–31, 2017.

[24] Xi Hang Cao, Ivan Stojkovic, and Zoran Obradovic. A robust data scaling algorithm to improve classification accuracies in biomedical data. *BMC bioinformatics*, 17(1):359, 2016.

[25] José María Cavanillas, Edward Curry, and Wolfgang Wahlster. The big data value opportunity. In *New horizons for a data-driven economy*, pages 3–11. Springer, Cham, 2016.

[26] Susan E Celniker and Gerald M Rubin. The drosophila melanogaster genome. *Annual review of genomics and human genetics*, 4(1):89–117, 2003.

[27] S Chandrasekar, R Dakshinamurthy, PG Seshakumar, B Prabavathy, and Chitra Babu. A novel indexing scheme for efficient handling of small files in hadoop distributed file system. In *2013 International Conference on Computer Communication and Informatics*, pages 1–8. IEEE, 2013.

[28] Guoqiang Jerry Chen, Janet L Wiener, Shridhar Iyer, Anshul Jaiswal, Ran Lei, Nikhil Simha, Wei Wang, Kevin Wilfong, Tim Williamson, and Serhat Yilmaz. Realtime data processing at facebook. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1087–1098, 2016.

[29] Trieu C Chieu and Hoi Chan. Dynamic resource allocation via distributed decisions in cloud environment. In *2011 IEEE 8th International Conference on e-Business Engineering*, pages 125–130. IEEE, 2011.

[30] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, et al. Web services description language (wsdl) 1.1, 2001.

[31] Cisco Systems. Cisco Visual Networking Index: Global Mobile data Traffic Forecast Update, 2013-2018. *White Papers*, 2014.

[32] Microsoft Corp. Azure Autoscale. https://azure.microsoft.com/en-us/features/autoscale/, 2019. [Online; accessed 11-01-2020].

[33] Intel Corporation. OpenCV. http://opencv.org, 2018. [Online; accessed 17-05-2020].

[34] International Business Machines Corporation. IBM BigInsight. http://www-01.ibm.com/software/data/infosphere/biginsights/quick-start/downloads.html, 2015. [Online; accessed 01-03-2015].

[35] Oracle Corporation. An Enterprise Architect's Guide to Big Data: Reference Architecture Overview. https://www.oracle.com/technetwork/topics/entarch/articles/oea-big-data-guide-1522052.pdf, 2016. [Online; accessed 18-05-2020].

[36] Tom Coughlin. 175 zettabytes by 2025. https://www.forbes.com/sites/tomcoughlin/2018/11/27/175-zettabytes-by-2025/#3ad581754597, 2018. [Online; accessed 17-05-2020].

[37] EGI Council. EGI. European Grid Infrastructure. https://www.egi.eu, 2001. [Online; accessed 17-05-2020].

[38] Julián Ramos Cózar, Nicolas Guil, José María González-Linares, Emilio L Zapata, and E Izquierdo. Logotype detection to support semantic-based video annotation. *Signal Processing: Image Communication*, 22(7):669–679, 2007.

[39] Julián Ramos Cózar, Pablo Nieto, José María González-Linares, Nicolas Guil, and Y Hernández-Heredia. Detection of logos in low quality videos. In *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, pages 630–635. IEEE, 2011.

[40] CSCE. PDPTA, Parallel and Distributed Processing Techniques and Applications. http://www.wikicfp.com/cfp/program?id=2332&s=PDPTA, 2017. [Online; accessed 18-05-2020].

[41] Mário David, Gonçalo Borges, Jorge Gomes, João Pina, Isabel Campos Plasencia, Enol Fernández-del Castillo, Iván Díaz, Carlos Fernandez, Esteban Freire, Álvaro Simón, Kostas Koumantaros, Michel Dreschner, Tiziana Ferrari, and Peter Solagna. Validation of Grid Middleware for the European Grid Infrastructure. *Journal of Grid Computing*, 12(3):543–558, sep 2014. ISSN 15729184. doi: 10.1007/s10723-014-9301-z.

[42] Jonathon Hare David Dupplaw and Sina Samangooei. OpenIMAJ. http://openimaj.org, 2014. [Online; accessed 18-05-2020].

[43] Infraestrutura Nacional de Computação Distribuída. INCD. https://www.incd.pt, 2020. [Online; accessed 17-05-2020].

[44] Escuela Internacional de Doctorado de la Universidad Rey Juan Carlos. *Libro de actas del I Congreso de la Escuela Internacional de Doctorado de la Universidad Rey Juan Carlos*, volume 1. Servicio de Publicaciones de la Universidad Rey Juan Carlos, 2018. [online] https://eciencia.urjc.es/handle/10115/16128.

[45] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Google,Inc.*, 2004.

[46] Patrick Debois. Agile 2008 toronto: Agile infrastructure and operations presentation. *URL: http://www. jedi. be/blog/2008/10/09/agile-2008-torontoagile-infrastructure-and-operations-presentation/. Accessed September*, 15:2017, 2008.

[47] Zhenyun Deng, Xiaoshu Zhu, Debo Cheng, Ming Zong, and Shichao Zhang. Efficient knn classification algorithm for big data. *Neurocomputing*, 195:143–148, 2016.

[48] Anand Deshpande and Manish Kumar. *Artificial Intelligence for Big Data: Complete Guide to Automating Big Data Solutions Using Artificial Intelligence Techniques*. Packt Publishing Ltd, 2018.

[49] Sihao Ding, Gang Li, Ying Li, Xinfeng Li, Qiang Zhai, Adam C Champion, Junda Zhu, Dong Xuan, and Yuan F Zheng. Survsurf: human retrieval on large surveillance video data. *Multimedia Tools and Applications*, 76(5):6521–6549, 2017.

[50] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. In *Present and ulterior software engineering*, pages 195–216. Springer, 2017.

[51] WIPO Economics and Statistics Series. World intellectual property indicators 2016. *WIPO publication*, 1(941E), 2016.

[52] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[53] Adil Fahad, Najlaa Alshatri, Zahir Tari, Abdullah Alamri, Ibrahim Khalil, Albert Y Zomaya, Sebti Foufou, and Abdelaziz Bouras. A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE transactions on emerging topics in computing*, 2(3):267–279, 2014.

[54] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z Broder. Summary cache: A scalable wide-area web cache sharing protocol. In *ACM SIGCOMM Computer Communication Review*, pages 254–265. ACM, 1998.

[55] Tongke Fan. Research and implementation of user clustering based on mapreduce in multimedia big data. *Multimedia Tools and Applications*, 77(8):10017–10031, 2018.

[56] Centro Investigacion Principe Felipe. High Performance Genomics Aligner. http://docs.bioinfo.cipf.es/projects/hpg-aligner, 2013. [Online; accessed January-2013].

[57] Eugen Feller, Lavanya Ramakrishnan, and Christine Morin. Performance and energy efficiency of big data applications in cloud environments: A hadoop case study. *Journal of Parallel and Distributed Computing*, 79:80–89, 2015.

[58] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. In *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*, pages 171–172. IEEE, 2015.

[59] Mike Ferguson. Architecting a big data platform for analytics. *A Whitepaper prepared for IBM*, 30, 2012. [online] https://essextec.com/wp-content/uploads/2015/09/PureData-Intelligent-Business-Strategies-Whitepaper.pdf.

[60] Alberto Fernandez, Ruben Casado, and Ruben Usamentiaga. A real-time big data architecture for glasses detection using computer vision techniques. In *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, pages 591–596. IEEE, 2015.

[61] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 390–398. IEEE, 2000.

[62] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[63] Open Grid Forum. OGF. https://www.ogf.org, 2006. [Online; accessed 17-05-2020].

[64] Ian Foster. The physiology of the grid: An open grid services architecture for distributed systems integration. *Globus Project*, 2002.

[65] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, 2003.

[66] Ian Foster, Carl Kesselman, et al. The grid: Blueprint for a new computing infrastructure. *Morgan Kaufmann Publishers Inc.: San Francisco, CA*, 677: 200–222, 1999.

[67] Ian Foster, Tom Maguire, and David Snelling. Ogsa wsrf basic profile 1.0. *Recommendation GWD-R GGF, OGSA WG, September*, 7, 2005.

[68] R Foundation. R-project. http://cran.r-project.org/, 2015. [Online; accessed 17-05-2020].

[69] Ákos Frohner, Jean-Philippe Baud, Rosa Maria Garcia Rioja, Gilbert Grosdidier, Rémi Mollon, David Smith, and Paolo Tedesco. Data management in egee. In *Journal of Physics: Conference Series*, page 062012. IOP Publishing, 2010.

[70] Apache Software Fundation. Apache Hadoop. http://hadoop.apache.org, 2018. [Online; accessed 17-05-2020].

[71] Apache Software Fundation. Apache Mesos. http://mesos.apache.org, 2018. [Online; accessed 11-01-2020].

[72] Apache Software Fundation. Apache Licences. https://www.apache.org/licenses, 2020. [Online; accessed 17-05-2020].

[73] Apache Software Fundation. Apache Cassandra. https://cassandra.apache.org, 2020. [Online; accessed 17-05-2020].

[74] Apache Software Fundation. Apache Flink. http://flink.apache.org, 2020. [Online; accessed 17-05-2020].

[75] Apache Software Fundation. Apache Hbase. https://hbase.apache.org, 2020. [Online; accessed 17-05-2020].

[76] Apache Software Fundation. HDFS Hadoop Distributed File System. https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html, 2020. [Online; accessed 17-05-2020].

[77] Apache Software Fundation. Apache Hive. https://hive.apache.org, 2020. [Online; accessed 17-05-2020].

[78] Apache Software Fundation. Apache Kafka. https://kafka.apache.org, 2020. [Online; accessed 17-05-2020].

[79] Apache Software Fundation. Apache Pig. https://pig.apache.org, 2020. [Online; accessed 17-05-2020].

[80] Apache Software Fundation. Apache Samza. http://samza.apache.org, 2020. [Online; accessed 17-05-2020].

[81] Apache Software Fundation. Apache Spark. http://spark.apache.org, 2020. [Online; accessed 17-05-2020].

[82] Apache Software Fundation. Apache Sqoop. https://sqoop.apache.org, 2020. [Online; accessed 17-05-2020].

[83] Apache Software Fundation. Apache Storm. http://storm.apache.org, 2020. [Online; accessed 17-05-2020].

[84] Apache Software Fundation. Apache Storm Trident. https://storm.apache.org/releases/current/Trident-tutorial.html, 2020. [Online; accessed 17-05-2020].

[85] Apache Software Fundation. Apache Zookeeper. https://zookeeper.apache.org, 2020. [Online; accessed 17-05-2020].

[86] Kubernetes Linux Fundation. Kubernetes performance Measurements and Roadmap. https://kubernetes.io/blog/2015/09/kubernetes-performance-measurements-and, 2015. [Online; accessed 11-01-2020].

[87] Krishna Gadey. Real-time analytics at pinterest. *Online article, February.*, page 1, 2015. [online] https://medium.com/pinterest-engineering/real-time-analytics-at-pinterest-1ef11fdb1099.

[88] Amir Gandomi and Murtaza Haider. Beyond the hype: Big data concepts, methods, and analytics. *International journal of information management*, 35 (2):137–144, 2015.

[89] John Gantz and David Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future*, 2007:1–16, 2012.

[90] Martin Gardner. Mathematical games: The fantastic combinations of john conway's new solitaire game "life". *Scientific American*, 223(4):120–123, 1970.

[91] Dan Garlasu, Virginia Sandulescu, Ionela Halcu, Giorgian Neculoiu, Oana Grigoriu, Mariana Marinescu, and Viorel Marinescu. A big data implementation based on grid computing. In *2013 11th RoEduNet International Conference*, pages 1–4. IEEE, 2013.

[92] Adi Gaskell. Becoming a data driven organization. https://www.forbes.com/sites/adigaskell/2016/10/28/becoming-a-data-driven-organization/#4ddc0ce14121, 2016. [Online; accessed 17-05-2020].

[93] Fred George. Microservices architecture: A personal journey of discovery. https://www.slideshare.net/fredgeorge/micro-service-architecure, 2012. [Online; accessed 17-05-2020].

[94] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. Youtube traffic characterization: a view from the edge. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 15–28. ACM, 2007.

[95] Mert Onuralp Gökalp, Kerem Kayabay, Mohamed Zaki, Altan Koçyiğit, P Erhan Eren, and Andy Neely. Big-data analytics architecture for businesses: a comprehensive review on new open-source big-data tools. *Cambridge Service Alliance: Cambridge, UK*, 2017.

[96] Ekaterina Gonina, Gerald Friedland, Eric Battenberg, Penporn Koanantakool, Michael Driscoll, Evangelos Georganas, and Kurt Keutzer. Scalable multimedia content analysis on parallel platforms using python. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 10(2):18, 2014.

[97] Casey S Greene, Jie Tan, Matthew Ung, Jason H Moore, and Chao Cheng. Big data bioinformatics. *Journal of cellular physiology*, 229(12):1896–1900, 2014.

[98] UCSC Genome Bioinformatics group. Genome Reference Consortium Human Reference 37 HG19 . http://hgdownload.cse.ucsc.edu/goldenPath/hg19/bigZips/, 2000. [Online; accessed 17-05-2020].

[99] Vipin Gupta, Karamjeet Kaur, and Sukhveer Kaur. Performance comparison between light weight virtualization using docker and heavy weight virtualization. *International Journal of Advanced Technology in Engineering and Science, Volume*, 1(05):509–514, 2017.

[100] Walid A Hanafy, Amr E Mohamed, and Sameh A Salem. A new infrastructure elasticity control algorithm for containerized cloud. *IEEE Access*, 7:39731–39741, 2019.

[101] HashiCorp. The Million Container Challenge. https://www.hashicorp.com/c1m, 2019. [Online; accessed 11-01-2020].

[102] Hossein Hassani and Emmanuel Sirimal Silva. Forecasting with big data: A review. *Annals of Data Science*, 2(1):5–19, 2015.

[103] Brenner Heintz and Denny Lee. Productionizing Machine Learning with Delta Lake. https://databricks.com/blog/2019/08/14/productionizing-machine-learning-with-delta-lake.html, 2019. [Online; accessed 17-05-2020].

[104] Cal Henderson. *Building scalable web sites*. O'Reilly Media, Inc., 2006.

[105] José Herrera and Ignacio Blanquer. Studying the improving of data locality on distributed grid applications in bioinformatics. In *7th IBERIAN Grid Infrastructure Conference Proceedings*, page 103, 2013.

[106] José Herrera and Germán Moltó. Detecting events in streaming multimedia with big data techniques. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 345–349. IEEE, 2016.

[107] José Herrera and Germán Moltó. Toward bio-inspired auto-scaling algorithms: An elasticity approach for container orchestration platforms. *IEEE Access*, 8: 52139–52150, 2020.

[108] José Herrera, Germán Moltó, and Atsuhiro Takasu. Logotype detection in streaming multimedia using apache storm. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pages 81–87. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2017.

[109] José Herrera. Cobeats - container bio-inspired enhanced autoscaling system. https://github.com/grycap/cobeats, 2018. [Online; accessed 11-01-2020].

[110] Philipp Hoenisch, Ingo Weber, Stefan Schulte, Liming Zhu, and Alan Fekete. Four-fold auto-scaling on a contemporary deployment platform using docker containers. In *International Conference on Service-Oriented Computing*, pages 316–323. Springer, 2015.

[111] Nils Homer, Barry Merriman, and Stanley F Nelson. Bfast: an alignment tool for large scale genome resequencing. *PloS one*, 4(11), 2009.

[112] Nils Homere. BFAST: Blat-like Fast Accurate Search Tool. http://sourceforge.net/apps/mediawiki/bfast/, 2009. [Online; accessed 17-05-2020].

[113] Han Hu, Yonggang Wen, Tat-Seng Chua, and Xuelong Li. Toward scalable systems for big data analytics: A technology tutorial. *IEEE Access*, 2:652–687, 2014.

[114] Eduardo Huedo, Rubén S Montero, and Ignacio M Llorente. A recursive architecture for hierarchical grid resource management. *Future Generation Computer Systems*, 25(4):401–405, 2009.

[115] Ali Hussain. Performance of Docker vs VMs. https://es.slideshare.net/Flux7Labs/performance-of-docker-vs-vms, 2014. [Online; accessed 11-01-2020].

[116] Forrest N Iandola, Anting Shen, Peter Gao, and Kurt Keutzer. Deeplogo: Hitting logo recognition with the deep neural network hammer. *arXiv preprint arXiv:1510.02131*, 2015.

[117] IEEE. IEEE Access. https://ieeeaccess.ieee.org/, 2020. [Online; accessed 18-05-2020].

[118] Dong-Hyuck Im, Cheol-Hye Cho, and IlGu Jung. Detecting a large number of objects in real-time using apache storm. In *Information and Communication Technology Convergence (ICTC), 2014 International Conference on*, pages 836–838. IEEE, 2014.

[119] Alphabet Inc. YouTube Statistics. https://www.youtube.com/yt/press/statistics.html, 2016. [Online; accessed 12-11-2016].

[120] Amazon Inc. Amazon Step Scaling. https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scaling-simple-step.html, 2018. [Online; accessed 11-01-2020].

[121] Amazon Inc. ECS Auto-Scaling. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-auto-scaling.html, 2019. [Online; accessed 11-01-2020].

[122] Docker Inc. Docker. https://www.docker.com/, 2020. [Online; accessed 18-05-2020].

[123] Euromicro Inc. Euromicro. https://www.euromicro.org/, 2020. [Online; accessed 18-05-2020].

[124] MongoDB Inc. MongoDB. https://www.mongodb.com, 2019. [Online; accessed 17-05-2020].

[125] Iberian Grid Infrastructure. Ibergrid. https://wibergrid.lip.pt/site, 2006. [Online; accessed 17-05-2020].

[126] Anil K Jain and Aditya Vailaya. Image retrieval using color and shape. *Pattern recognition*, 29(8):1233–1244, 1996.

[127] Greg Jarboe. VidCon 2015 Haul: Trends, Strategic Insights, Critical Data, and Tactical Advice. http://tubularinsights.com/vidcon-2015-strategic-insights-tactical-advice/, 2015. [Online; accessed 17-05-2020].

[128] Jelastic. Jelastic Web site. https://jelastic.com, 2019. [Online; accessed 11-01-2020].

[129] Dejun Jiang, Guillaume Pierre, and Chi-Hung Chi. Autonomous resource provisioning for multi-service web applications. In *Proceedings of the 19th international conference on World wide web*, pages 471–480. ACM, 2010.

[130] Godson Koffi Kalipe and Rajat Kumar Behera. Big data architectures: A detailed and application oriented review. *International Journal of Innovative Technology and Exploring Engineering*, 2019.

[131] Chuanqi Kan. Docloud: An elastic cloud platform for web applications based on docker. In *Advanced Communication Technology (ICACT), 2016 18th International Conference on*, pages 478–483. IEEE, 2016.

[132] Arpan Kumar Kar. Bio inspired computing–a review of algorithms and scope of applications. *Expert Systems with Applications*, 59:20–32, 2016.

[133] Anil Karmel, Ramaswamy Chandramouli, and Michaela Iorga. Nist definition of microservices, application containers and system virtual machines. Technical report, National Institute of Standards and Technology, 2016.

[134] Pankaj Deep Kaur and Inderveer Chana. A resource elasticity framework for qos-aware execution of cloud applications. *Future Generation Computer Systems*, 37:14–25, 2014.

[135] Anastasios Kesidis and Dimosthenis Karatzas. Logo and trademark recognition. In *Handbook of Document Image Processing and Recognition*, pages 591–646. Springer, 2014.

[136] Rohit Khurana. *Software Engineering (WBUT), Fourth Ed.* Vikas Publishing House PVT Ltd., 2016.

[137] Myoungjin Kim, Yun Cui, Seungho Han, and Hanku Lee. Towards efficient design and implementation of a hadoop-based distributed video transcoding system in cloud computing environment. *International Journal of Multimedia and Ubiquitous Engineering*, 8(2):213–224, 2013.

[138] Kim, M. and Han,S. and Cui,Y. and Jeong, C. A Hadoop-based Multimedia Transcoding System for Procesing Social Media in the PaaS Platform of SMCCSE. *KSII Transactios on Internet and Information Systems*, 2012. http://dcslab.konkuk.ac.kr/papers/TIIS_Vol6No11P5Nov2012.pdf.

[139] Mariam Kiran, Peter Murphy, Inder Monga, Jon Dugan, and Sartaj Singh Baveja. Lambda architecture for cost-effective batch and speed big data processing. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 2785–2792. IEEE, 2015.

[140] Andrzej Kochut and Kirk Beaty. On strategies for dynamic resource management in virtualized server environments. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2007. MASCOTS'07. 15th International Symposium on*, pages 193–200. IEEE, 2007.

[141] Ricardo Koller and Raju Rangaswami. I/o deduplication: Utilizing content similarity to improve i/o performance. *ACM Transactions on Storage (TOS)*, 6 (3):1–26, 2010.

[142] Joanna Kołodziej and Samee Ullah Khan. Data scheduling in data grids and data centers: a short taxonomy of problems and intelligent resolution techniques. In *Transactions on Computational Collective Intelligence X*, pages 103–119. Springer, 2013.

[143] Bartosz Konieczny. Zeta architecture. https://www.waitingforcode.com/general-big-data/zeta-architecture/read, 2015. [Online; accessed 17-05-2020].

[144] Jay Kreps. Questioning the lambda architecture. *Online article, July*, page 205, 2014. [online] https://www.oreilly.com/radar/questioning-the-lambda-architecture.

[145] Johannes Kroß, Andreas Brunnert, Christian Prehofer, Thomas A Runkler, and Helmut Krcmar. Stream processing on demand for lambda architectures. In *European Workshop on Performance Engineering*, pages 243–257. Springer, 2015.

[146] Priyanka P Kukade and Geetanjali Kale. Auto-scaling of micro-services using containerization. *International Journal of Science and Research (IJSR)*, 4(9): 1960–1963, 2015.

[147] Anuj Kumar. *Architecting data-intensive applications*. Packt Publishing, 2018.

[148] Lawrence Berkeley National Laboratory. NASA Kennedy Space Center WWW server HTTP requests. ftp://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html, 1995. [Online; accessed 11-01-2020].

[149] Lawrence Berkeley National Laboratory. 1998 World Cup Web site requests between April 30, 1998 and July 26, 1998. ftp://ita.ee.lbl.gov/html/contrib/WorldCup.html, 1998. [Online; accessed 11-01-2020].

[150] Doug Laney. 3d data management: Controlling data volume, velocity, and variety. application delivery strategies. *Recuperado de http://blogs. gartner. com/doug-laney/files/2012/01/ad949-3D-Data-Management-Con-trolling-Data-Volume-Velocity-and-Variety. pdf*, 2001.

[151] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome biology*, 10(3):R25, 2009.

[152] Steve LaValle, Eric Lesser, Rebecca Shockley, Michael S Hopkins, and Nina Kruschwitz. Big data, analytics and the path from insights to value. *MIT sloan management review*, 52(2):21–32, 2011.

[153] Denny Lee. Beyond Lambda: Introducing Delta Architecture. https://www.youtube.com/watch?v=FePv0lro0z8, 2020. [Online; accessed 17-05-2020].

[154] In Lee. Big data: Dimensions, evolution, impacts, and challenges. *Business Horizons*, 60(3):293–303, 2017.

[155] J. Leibiusky, G. Eisbruch, and D. Simonassi. *Getting Started with Storm*. Oreilly and Associate Series. O'Reilly Media, 2012. ISBN 9781449324018. URL https://books.google.co.jp/books?id=s-_ovXmcrIoC.

[156] James Lewis. Microservices-java, the unix way. In *Proceedings of the 33rd Degree Conference for Java Masters*, 2012.

[157] James Lewis and Martin Fowler. Microservices. *MartinFowler. com*, 2014. [online] https://martinfowler.com/articles/microservices.html.

[158] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *bioinformatics*, 25(14):1754–1760, 2009.

[159] Ruiqiang Li, Chang Yu, Yingrui Li, Tak-Wah Lam, Siu-Ming Yiu, Karsten Kristiansen, and Jun Wang. Soap2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

[160] Google LLC. Kubernetes. https://kubernetes.io/, 2018. [Online; accessed 18-05-2020].

[161] Google LLC. Google Auto-Scaling. https://cloud.google.com/compute/docs/autoscaler/, 2019. [Online; accessed 11-01-2020].

[162] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of grid computing*, 12(4):559–592, 2014.

[163] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[164] Andrea Luzzardi. Scale testing Docker Swarm to 30,000 Containers. https://blog.docker.com/2015/11/scale-testing-docker-swarm-30000-containers/, 2015. [Online; accessed 11-01-2020].

[165] C Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, Rebekah Metz, and Booz Allen Hamilton. Reference model for service oriented architecture 1.0. *OASIS standard*, 12(S 18), 2006.

[166] Ken MacLean. Voxforge Spanish Webpage. http://www.voxforge.org/es, 2015. [Online; accessed 17-05-2020].

[167] James Manyika. Big data: The next frontier for innovation, competition, and productivity. *http://www. mckinsey. com/Insights/MGI/Research/Technology_and_Innovation/Big_data_The_next_frontier_for_innovation*, 2011.

[168] Inc MAPR Technologiesl. Building the data-centric enterprise. *MAPR White Paper*, 2015. [online] https://mapr.com/whitepapers/data-centric-enterprise/assets/data-centric-enterprise.pdf.

[169] Diego García-Saiz Marta Zorrilla. Arquitecturas y tecnologías para el big data. https://ocw.unican.es/pluginfile.php/2396/course/section/2473/tema%203.2%20Arquitecturas%20y%20tecnologi%CC%81as%20para%20el%20big%20data.pdf, 2017. [Online; accessed 18-05-2020].

[170] Nathan Marz. How to beat the cap theorem. http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html, 2011. [Online; accessed 17-05-2020].

[171] Nathan Marz and James Warren. *Big data: principles and best practices of scalable real-time data systems*. Manning Publications Co., 2013.

[172] Matt McNaughton. *Predictive Pod Auto-scaling in the Kubernetes Container Cluster Manager*. PhD thesis, Williams College, Williamstown, Massachusetts, 2016. unpublished thesis.

[173] PrathyushaRani Merla and Yiheng Liang. Data analysis using hadoop mapreduce environment. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 4783–4785. IEEE, 2017.

[174] Valter Rogério Messias, Julio Cezar Estrella, Ricardo Ehlers, Marcos José Santana, Regina Carlucci Santana, and Stephan Reiff-Marganiec. Combining time series prediction models using genetic algorithm to autoscaling web applications hosted in the cloud infrastructure. *Neural Computing and Applications*, 27(8):2383–2406, 2016.

[175] Jeffrey Mogul, Balachander Krishnamurthy, Fred Douglis, Anja Feldmann, Yaron Goland, Arthur van Hoff, and D Hellerstein. Delta encoding in http. *IETF, Gennaio*, 65, 2002.

[176] Germán Moltó, Miguel Caballer, and Carlos de Alfonso. Automatic memory-based vertical elasticity and oversubscription on cloud platforms. *Future Generation Computer Systems*, 56:1–10, mar 2016. ISSN 0167-739X. doi: 10.1016/j.future.2015.10.002. URL http://dx.doi.org/10.1016/j.future.2015.10.002http://linkinghub.elsevier.com/retrieve/pii/S0167739X15003155.

[177] Laura R Moore, Kathryn Bean, and Tariq Ellahi. Transforming reactive auto-scaling into proactive auto-scaling. In *Proceedings of the 3rd International Workshop on Cloud Data and Platforms*, pages 7–12. ACM, 2013.

[178] Adrian Mouat. *Using Docker: Developing and Deploying Software with Containers*. O'Reilly Media, Inc., 2016. ISBN 9781491915769.

[179] Francesc D Muñoz-Escoí and José M Bernabéu-Aubán. A survey on elasticity management in paas systems. *Computing*, 99(7):617–656, 2017.

[180] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, and Mike Amundsen. *Microservice architecture: aligning principles, practices, and culture*. O'Reilly Media, Inc., 2016.

[181] Maryam M Najafabadi, Flavio Villanustre, Taghi M Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. Deep learning applications and challenges in big data analytics. *Journal of Big Data*, 2(1):1, 2015.

[182] Amro Najjar, Xavier Serpaggi, Christophe Gravier, and Olivier Boissier. Multi-agent negotiation for user-centric elasticity management in the cloud. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 357–362. IEEE Computer Society, 2013.

[183] Sam Newman. *Building microservices: designing fine-grained systems*. O'Reilly Media, Inc., 2015.

[184] Jeff Nickoloff. Evaluating Container Platforms at Scale. https://medium.com/on-docker/evaluating-container-platforms-at-scale-5e7b44d93f2c, 2016. [Online; accessed 11-01-2020].

[185] OpenShift. Best Practices for Horizontal Application Scaling. https://blog.openshift.com/best-practices-for-horizontal-application-scaling/, 2013. [Online; accessed January-2018].

[186] Soumaya Ounacer, Mohamed Amine, Soufiane Ardchir, Abderrahmane Daif, and Mohamed Azouazi. A New Architecture for Real Time Data Stream Processing. *International Journal of Advanced Computer Science and Applications*, 8(11), 2017. ISSN 2158107X. doi: 10.14569/ijacsa.2017.081106. URL www.ijacsa.thesai.org.

[187] Jyoti Parsola, Durgaprasad Gangodkar, and Ankush Mittal. Efficient storage and processing of video data for moving object detection using hadoop/mapreduce. In *Proceedings of the International Conference on Signal, Networks, Computing, and Systems*, pages 137–147. Springer, 2017.

[188] Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.

[189] Hsiao-Lin Peng and Shu-Yuan Chen. Trademark shape recognition using closed contours. *Pattern Recognition Letters*, 18(8):791–803, 1997.

[190] Rafael Pereira, Marcello Azambuja, Karin Breitman, and Markus Endler. An architecture for distributed high performance video processing in the cloud. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 482–489. IEEE, 2010.

[191] Salvatore Pinto. Grid Cache Server. http://sourceforge.net/p/gridcacheserver/home/Home/, 2009. [Online; accessed 17-05-2020].

[192] CERN. Organisation Européenne pour la Recherche Nucléaire. ATLAS Experiment. A Toroidal LHC ApparatuS. https://atlas.cern/, 1998. [Online; accessed 17-05-2020].

[193] CERN. Organisation Européenne pour la Recherche Nucléaire. CMS Experiment. Compact Muon Solenoid. http://cms.cern, 1998. [Online; accessed 17-05-2020].

[194] CERN. Organisation Européenne pour la Recherche Nucléaire. LHC. Large Hadron Collider. https://home.cern/science/accelerators/large-hadron-collider, 1998. [Online; accessed 17-05-2020].

[195] CERN. Organisation Européenne pour la Recherche Nucléaire. gLite-Lightweight Middleware for Grid Computing. http://grid-deployment.web.cern.ch/grid-deployment/glite-web/, 2004. [Online; accessed 17-05-2020].

[196] CERN. Organisation Européenne pour la Recherche Nucléaire. GFAL2. Grid File Access Library. https://dmc.web.cern.ch/projects/gfal-2/home, 2014. [Online; accessed 17-05-2020].

[197] CERN. Organisation Européenne pour la Recherche Nucléaire. CERNVM CernVM File System. http://cernvm.cern.ch/portal/filesystem, 2020. [Online; accessed 17-05-2020].

[198] Chenhao Qu, Rodrigo N Calheiros, and Rajkumar Buyya. Auto-scaling web applications in clouds: A taxonomy and survey. *arXiv preprint arXiv:1609.09224*, 2016.

[199] Babak Bashari Rad, Harrison John Bhatti, and Mohammad Ahmadi. An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS)*, 17(3):228, 2017.

[200] M Mazhar Rathore, Hojae Son, Awais Ahmad, Anand Paul, and Gwanggil Jeon. Real-time big data stream processing using gpu with spark over hadoop ecosystem. *International Journal of Parallel Programming*, 46(3):630–646, 2018.

[201] David Reinsel, John Gantz, and John Rydning. The digitization of the world from edge to core. *IDC White Paper*, 2018. [online] https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf.

[202] Leonard Richardson and Sam Ruby. *RESTful web services*. O'Reilly Media, Inc., 2008.

[203] Ignacio Riquelme Medina. *Revisión de los Algoritmos Bioinspirados*. PhD thesis, University of Manchester, 07 2014.

[204] Fabiana Rossi, Matteo Nardelli, and Valeria Cardellini. Horizontal and vertical scaling of container-based applications using reinforcement learning. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 329–338. IEEE, 2019.

[205] Mickael Rouvier, Grégor Dupuy, Paul Gay, Elie Khoury, Teva Merlin, and Sylvain Meignier. An open-source state-of-the-art toolbox for broadcast news diarization. In *InterSpeech*, 2013.

[206] Dominik Ryzko. *Modern Big Data Architectures: A Multi-Agent Systems Perspective*. John Wiley & Sons, 2020.

[207] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

[208] Marcel Salathe. *Nature in Code*. Leanpub, 2016. [online] http://www.natureincode.com/code/various/ants.html.

[209] S.L. Salzberg, D.B. Searls, and S. Kasif. *Computational Methods in Molecular Biology*. New comprehensive biochemistry. Elsevier, 1999. ISBN 9780444502049.

[210] Iman Samizadeh. A brief introduction to two data processing architectures—lambda and kapp. for big data. *March https://towardsdatascience. com*, 2018.

[211] Bertil Schmidt and Andreas Hildebrandt. Next-generation sequencing: big data meets high performance computing. *Drug discovery today*, 22(4):712–717, 2017.

[212] Rainer Schmidt and Matthias Rella. An approach for processing large and non-uniform media objects on mapreduce-based clusters. In *Digital Libraries: For Cultural Heritage, Knowledge Dissemination, and Future Creation*, pages 172–181. Springer, 2011.

[213] Guido Schmutz. Last week in Stream Processing & Analytics 4/18/2016. https://guidoschmutz.wordpress.com/2016/04/20/last-week-in-stream-processing-analytics-4182016, 2016. [Online; accessed 18-05-2020].

[214] Klaus Schwab. *The fourth industrial revolution*. Currency, 2017.

[215] Jim Scott. Zeta Architecture: Hexagon is the new circle. https://www.oreilly.com/content/zeta-architecture-hexagon-is-the-new-circle/, 2015. [Online; accessed 17-05-2020].

[216] Cynthia M. Saracco y Scott Lindner Seeling Cheung, Luciano Resende. Desarrollo de una Aplicación de Big Data para Explorar y Descubrir Datos. https://www.ibm.com/developerworks/ssa/data/library/app_bigdata/872864.html, 2013. [Online; accessed 18-05-2020].

[217] Pavel Senin. Dynamic time warping algorithm review. *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, 855(1-23): 40, 2008.

[218] Saeed Shahrivari. Beyond batch processing: towards real-time and streaming big data. *Computers*, 3(4):117–129, 2014.

[219] Farzana Shaikh, Danish Pawaskar, Abutalib Siddiqui, and Umar Khan. Youtube data analysis using mapreduce on hadoop. In *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pages 2037–2041. IEEE, 2018.

[220] Konstantin Shvachko. Hadoop-1687 Name-node memory size estimates and optimazation proposal. https://issues.apache.org/jira/browse/HADOOP-1687, 2009. [Online; accessed 17-05-2020].

[221] Kamakhya Narain Singh, Rajat Kumar Behera, and Jibendu Kumar Mantri. Big data ecosystem: Review on architectural evolution. In *Emerging Technologies in Data Mining and Information Security*, pages 335–345. Springer, 2019.

[222] Seokho Son and Kwang Mong Sim. A price-and-time-slot-negotiation mechanism for cloud service reservations. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(3):713–728, 2012.

[223] Heinz Stockinger. Defining the grid: a snapshot on the current view. *The Journal of Supercomputing*, 42(1):3–17, 2007.

[224] Saravanan Subramanian. Features of Apache Big Data Streaming Frameworks. https://dzone.com/articles/features-of-apache-big-data-streaming-frameworks, 2016. [Online; accessed 18-05-2020].

[225] Nicoleta Tantalaki, Stavros Souravlas, and Manos Roumeliotis. A review on big data real-time stream processing and its scheduling techniques. *International Journal of Parallel, Emergent and Distributed Systems*, pages 1–31, 2019.

[226] Sudeep Tanwar, Sudhanshu Tyagi, and Neeraj Kumar. *Multimedia Big Data Computing for IoT Applications: Concepts, Paradigms and Solutions*, volume 163. Springer, 2019.

[227] Neo Technology. Neo4j. https://neo4j.com, 2017. [Online; accessed 17-05-2020].

[228] Jose Salavert Torres, Ignacio Blanquer Espert, Andres Tomas Dominguez, Vicente Hernendez, Ignacio Medina, Joaquin Terraga, and Joaquin Dopazo. Using gpus for the exact alignment of short-read genetic sequences by means of the burrows-wheeler transform. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(4):1245–1256, 2012.

[229] Carnegie Mellon University. CMU Sphinx. Open source speech recognition toolkit. https://cmusphinx.github.io, 2000. [Online; accessed 18-05-2020].

[230] Nantes Angers Le Mans University. LIUM_spkDiarization Webpage. https://projets-lium.univ-lemans.fr/spkdiarization/, 2015. [Online; accessed 17-05-2020].

[231] P Ushapreethi, Balajee Jeyakumar, and P BalaKrishnan. Action recongnition in video survillance using hipi and map reducing model. *International Journal of Mechanical Engineering and Technology*, 8(11):368–375, 2017.

[232] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, pages 1–16, 2013.

[233] John Von Neumann and Arthur Walter Burks. *Theory of self-reproducing automata*. University of Illinois Press Urbana, 1996.

[234] Hanli Wang, Yun Shen, Lei Wang, Kuangtian Zhufeng, Wei Wang, and Cheng Cheng. Large-scale multimedia data mining using mapreduce framework. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 287–292. IEEE, 2012.

[235] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, 2012. ISBN 1449311520. URL http://www.amazon.com/Hadoop-Definitive-Guide-Tom-White/dp/1449311520.

[236] Alex Woodie. Why Gartner dropped big data off the hype curve. https://www.datanami.com/2015/08/26/why-gartner-dropped-big-data-off-the-hype-curve, 2015. [Online; accessed 18-05-2020].

[237] Jinsong Wu, Song Guo, Jie Li, and Deze Zeng. Big data meet green challenges: Big data toward green applications. *IEEE Systems Journal*, 10(3):888–900, 2016.

[238] Wei-Qi Yan, Jun Wang, and Mohan S Kankanhalli. Automatic video logo detection and removal. *Multimedia Systems*, 10(5):379–391, 2005.

[239] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. Live video analytics at scale with approximation and delay-tolerance. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 377–392, 2017.

[240] Weishan Zhang, Pengcheng Duan, Qinghua Lu, and Xin Liu. A realtime framework for video object detection with storm. In *Ubiquitous Intelligence and Computing, 2014 IEEE 11th Intl Conf on and IEEE 11th Intl Conf on and Autonomic and Trusted Computing, and IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UTC-ATC-ScalCom)*, pages 732–737. IEEE, 2014.

[241] Weishan Zhang, Liang Xu, Pengcheng Duan, Wenjuan Gong, Qinghua Lu, and Su Yang. A video cloud platform combing online and offline cloud computing technologies. *Personal and Ubiquitous Computing*, 19(7):1099–1110, 2015.

[242] Weishan Zhang, Haoyun Sun, Dehai Zhao, Liang Xu, Xin Liu, Jiehan Zhou, Huansheng Ning, Yi Guo, and Su Yang. A streaming cloud platform for real-time video processing on embedded devices. *IEEE Transactions on Cloud Computing*, 2019.

Apendice A

Autorizaciones para la Inclusión de Publicaciones

UNIVERSITY OF GEORGIA

Department of Computer Science
415 Boyd Graduate Studies Research Center
200 D.W. Brooks Drive
University of Georgia
Athens, Georgia 30602-7404
TEL 706-542-2911 | FAX 706-542-2966
www.cs.uga.edu

Franklin College of Arts & Sciences
Department of Computer Science

April 12, 2020

Dear Jose Herrera:

Permission Granted for use/re-use and inclusion in UPV institutional repository and/or thesis of the following published paper: (in full or in part)

> *Herrera, J , Moltó G. and Takasu, A. "Logotype Detection in Streaming Multimedia Using Apache Storm" - 2017 Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*

Sincerely,

Hamid R. Arabnia, PhD.
Senior Editor, PDPTA books
Officer, CSREA Press
Chair, PDPTA Steering Committee
Professor Emeritus, Computer Science

Tel: (706) 542-3480 / Cell/Mobile: (706) 340-4707
URL: http://cobweb.cs.uga.edu/~hra/
email: hra@uga.edu

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA DE DOCTORAT

The doctoral thesis developed by *José Herrera Hernández*, with draft title "Advanced optimization of parallel processing for big data", is presented by compendium of articles.

Mr./Ms. Adsuhiro Takasu appears as co-author of the publication(s) listed below:

- Herrera, J , Moltó G. and Takasu, A. "Logotype Detection in Streaming Multimedia Using Apache Storm" 2017 Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)

And so declares his acceptance to the inclusion of this articles in the aforementioned doctoral thesis, together with his resignation to use the same article as part of another thesis.

Tokyo, April 14th, 2020

Signed: *Adsuhiro Takasu*

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA DE DOCTORAT

La tesis doctoral desarrollada por José Herrera Hernández, con título "**Optimización de arquitecturas distribuidas para el procesado de datos masivos**" se presenta por compendio de artículos.

Dr. Ignacio Blanquer Espert, figura como co-autor de la publicación:

Herrera, J. , Blanquer, I. "Studying the improving of data locality on distributed Grid applications in bioinformatics" , 7th IBERIAN Grid Infrastructure Conference Proceedings, 2013, pg. 103-115

Y declara su aceptación para que el doctorando presente el trabajo como parte de su tesis doctoral, a la vez que expresa su renuncia a presentar este mismo artículo como parte de otra tesis doctoral.

En Valencia, *a* 6 de mayo de 2020

IGNACIO| BLANQUER| ESPERT

Firmado digitalmente por IGNACIO| BLANQUER|ESPERT
Fecha: 2020.05.07 08:17:41 +02'00'

Firmado: Ignacio Blanquer Espert

Apendice B

Perfiles de los Congresos y Medios de Publicación

B.1 IberGrid

La Infraestructura de Grid Ibérica (IberGrid) [125] se inicia en 2007 bajo el marco del acuerdo bilateral para la Ciencia y la Tecnología que se firmó en noviembre de 2003 entre Portugal y España, con el fin de crear una infraestructura de Grid en la península y promover la cooperación entre ambos países en los campos de informática de Grid y supercomputación.

El objetivo principal del IberGrid es el de construir un foro donde los avances en el desarrollo de infraestructuras de Grid y Cloud, tecnologías y sus usos, sean compartidos por los actores principales en países ibéricos y latinoamericanos.

Las infraestructuras comunes son cruciales para solucionar muchos de los problemas y son cada vez más importantes en los negocios, atención de la salud, el medio ambiente y otros usos. Las conferencias IberGrid son una oportunidad excelente de congregar a una amplia comunidad de académicos, investigadores, estudiantes, especialistas de la industria y médicos en todas las ramas del conocimiento que comparten una necesidad común. Esta comunidad se beneficia de la infraestructura común ibérica, lo que facilitará el acceso a un cada vez más poderoso escenario de recursos distribuidos.

El acceso a la infraestructura de IberGrid se realiza mediante las correspondientes infraestructuras nacionales. En Portugal es el INCD (Infraestrutura Nacional de Computação Distribuída) y por España el ES-NGI (*National Grid Initiative* de España).

B.2 PDP

Euromicro [123] es una organización internacional científica y educativa dedicada al avance de las artes, ciencias y aplicaciones de la tecnología de la información y la microelectrónica. Fue fundada en 1973, inspirada en la tecnología emergente de microprocesadores. Desde entonces, Euromicro se ha dedicado a la promoción de la investigación y a la difusión de conocimientos e información.

Su objetivo principal es la organización de conferencias y talleres de microelectrónica, ciencias de la computación e ingeniería informática, siendo temas como la ingeniería del software, los sistemas de tiempo real, el procesamiento paralelo y distribuido, multimedia, telecomunicaciones y la arquitectura de computadores algunos de sus temas de interés.

Euromicro es miembro del Foro Estratégico de la Alianza Europea para la Innovación (EAI). Entre las actividades que desarrolla está una conferencia de sistemas en tiempo real (ERTS - Euromicro Conference on Real-Tiem Systems), PDP (Euromicro International Conference on Parallel, Distributed and Network-based Processing) o DSD/SEAA (Euromicro Digital System Design - Euromicro Conference in Software Engineering and Advanced Applications)

Las actas de sus congresos se publican a través de IEEE Computer Press. Euromicro con Elsevier Science dirige dos revistas científicas acreditadas: JSA (Embedded Software Design) y MICPRO (Embedded Hardware Design).

B.3 PDPTA

PDPTA (International Conference on Parallel and Distributed Processing Techniques and Applications) [40] está dentro del CSCE (Congress in Computer Science, Computer Engineering and Applied Computing) que organiza la *American Council on Science and Education*. El congreso, catalogado como Core B por el índice ERA 2010, se encuentra entre las cinco principales reuniones anuales de investigadores en informática, ingeniería informática y computación aplicada, reuniendo asistentes de aproximadamente 75 países.

B.4 IEEE Access

IEEE Access [117] es una revista científica de acceso abierto revisada por pares que publica el IEEE (Instituto de Ingenieros Eléctricos y Electrónicos). El factor de impacto

de la publicación para el año 2018 fue de 4.098 y fue clasificada en el primer cuartil (Q1) del JCR. Fue creada en 2013 con los mismos campos de interés de la IEEE. La revista ganó el premio PROSE en 2015 como mejor revista de ciencia, tecnología, ingeniería y matemáticas.

B.5 Congreso EID de la URJC

El Congreso de la Escuela Internacional de Doctorado de la Universidad Rey Juan Carlos [44], ha reunido en los dos últimos años a doctorandos y profesores para intercambiar experiencias fomentando el debate y la comunicación. Entre sus objetivos podemos encontrar la posibilidad de reflexionar sobre los problemas de los estudiantes de doctorado de diferentes áreas del conocimiento y la difusión de la actividad investigadora de todos los miembros de la comunidad universitaria.