



UNIVERSIDADE DA CORUÑA

Facultade de Informática

Máster Universitario en Enxeñaría Informática

TRABAJO FIN DE MÁSTER

**Análisis de necesidades de cómputo
intensivo y despliegue de un sistema de
colas para un CPD**

Estudiante: Mauro García Barro

Dirección: José Carlos Dafonte Vázquez

Javier Pereira Loureiro

A Coruña, junio de 2025.

Resumen

En un centro de procesamiento de datos resulta imprescindible maximizar el rendimiento y la optimización de los recursos, en especial cuando se ejecutan tareas de cómputo intensivo como el entrenamiento de un modelo de inteligencia artificial. Esta situación provoca que se dependa principalmente del uso de tarjetas gráficas para agilizar la compleción de dichas actividades. Sin embargo, en el Área de Sistemas del CITIC han sido afectados recientemente por cambios en los costes y políticas de licenciamiento en el uso de este tipo de componentes, por lo que se han visto en la obligación de explorar diversas alternativas de virtualización para asegurar la continuidad de sus proyectos de investigación. Dentro de este contexto los *job schedulers* cobran una gran importancia, ya que se encargan de mantener una designación versátil de los recursos de acuerdo a la situación del entorno en cada momento.

Así pues en este trabajo se expone, por un lado, el desarrollo de una propuesta de entorno basada en estos planificadores que ayude a resolver el problema de alojamiento de las peticiones y, por otro lado, la configuración del despliegue de varias plataformas de virtualización, estableciendo un contraste de su capacidad técnica y con ello reflexionando sobre si compensa migrar el resto de la infraestructura actual; todo ello desarrollado dentro del marco de la metodología de SCRUM.

Abstract

In a data processing center is essential to maximize performance and resource optimization, especially when intensive computing tasks such as training an artificial intelligence model are executed. This situation provokes a dependency of graphical cards in order to speed up the completion of such activities. However, CITIC's System Area has been affected recently by changes in costs and licensing policies for the use of this kind of components, forcing them to explore various virtualization alternatives to ensure the continuity of their research projects. Inside this context *job schedulers* reach a great importance, because they are designed to maintain a versatile resource allocation according to the environment situation at any given time.

Therefore this paper presents, on one hand, the development for an environment proposal based on these planners that helps solve the problem of request hosting and, on the other hand, the configuration for deploying several virtualization platforms, establishing a contrast of their technical capacity and thus reflecting on whether it is worth migrating the

rest of the current infrastructure; all of this developed within the framework of the SCRUM methodology.

Palabras clave:

- Planificación de trabajos
- Centro de datos
- Clúster
- Gestor de recursos
- Plataforma de virtualización
- Análisis comparativo
- SLURM
- Hyper-V
- Proxmox
- VMWare

Keywords:

- Job Scheduling
- Datacenter
- Cluster
- Resource Manager
- Virtualization platform
- Benchmark
- SLURM
- Hyper-V
- Proxmox
- VMWare

Índice general

1	Introducción	1
1.1	Objetivos	2
1.2	Estructura de la memoria	3
2	Conceptos previos	5
2.1	Qué es HPC	5
2.1.1	Modelos de computación	5
2.1.2	Tendencias actuales	6
2.2	Centros de datos	8
2.2.1	Evolución	8
2.2.2	Tipos de infraestructura	9
2.2.3	Acerca de la nube	9
2.3	Virtualización: técnicas y plataformas	10
2.4	Sistemas de gestión de recursos	11
2.4.1	Estrategias de programación	12
2.4.2	Relación con HPC	13
2.5	Benchmarking	13
3	Metodología y planificación	15
3.1	Metodología	15
3.1.1	Roles	16
3.1.2	Artefactos	17
3.1.3	Ceremonias	17
3.1.4	Justificación	18
3.2	Planificación	18
3.2.1	Definición del backlog	18
3.2.2	Estimación temporal	19
3.2.3	Estimación de costes	20

3.2.4	Seguimiento	21
3.2.5	Gestión de riesgos	21
3.3	Tecnologías a utilizar	24
3.3.1	Desarrollo del sistema de colas	24
3.3.2	Desarrollo de los virtualizadores	26
3.3.3	Desarrollo de los benchmarks	27
3.3.4	Otras herramientas	28
4	Despliegue del clúster	31
4.1	Análisis de la infraestructura actual	31
4.2	Requisitos previos	32
4.3	Diseño inicial y topología	32
4.4	Implementación	33
4.4.1	Usuarios y sincronización	33
4.4.2	Autenticación de comunicaciones	34
4.4.3	Almacenamiento compartido	35
4.4.4	Compilación del paquete	36
4.4.5	Instalación	37
4.5	Lanzamiento y pruebas	41
4.5.1	Comandos principales	42
4.5.2	Creación de una carga de trabajo	43
4.5.3	Escenario 1: Priorización por backfilling	44
4.5.4	Escenario 2: Limitación de recursos global	44
4.5.5	Escenario 3: Limitación de recursos por cuenta	45
4.5.6	Monitorización de las colas	46
5	Configuración de virtualizadores	49
5.1	Despliegue en Hyper-V	49
5.1.1	Incorporación del rol	49
5.1.2	Acceso a red	50
5.1.3	Creación de máquina virtual	50
5.2	Despliegue en Proxmox	53
5.2.1	Instalación de dependencias	54
5.2.2	Configuración de interfaz puente	55
5.2.3	Creación de máquina virtual	56
5.3	Despliegue en VMware ESXi	58
5.3.1	Definición de almacén de datos	58
5.3.2	Asociación de grupo de puertos	58

5.3.3	Creación de máquina virtual	59
5.3.4	Conexión de tarjetas gráficas por passthrough	60
6	Resultados obtenidos	63
6.1	Prueba de rendimiento	63
6.1.1	Métricas a considerar	64
6.1.2	Extracción y análisis de datos	64
6.2	Prueba de Tensorflow	69
6.3	Prueba de estrés de GPU	69
6.4	Desglose económico	70
6.5	Resumen del procedimiento	71
7	Conclusiones y trabajo futuro	73
7.1	Conclusiones	73
7.2	Líneas de trabajo futuro	75
A	Material adicional	79
A.1	Configuración de Slurm	79
A.2	Despliegue de Prometheus/Grafana	83
A.3	Benchmarks y cargas de trabajo	83
A.4	Extracción de resultados del benchmark de rendimiento	85
A.5	Propuesta de plan de migración a largo plazo	88
A.5.1	Alcance	88
A.5.2	Composición inicial	88
A.5.3	Planificación	89
A.5.4	Migración de prueba	89
A.5.5	Comprobaciones post-migración	90
Lista de acrónimos		91
Glosario		93
Bibliografía		95

Índice de figuras

2.1	Visualización de tipos de hipervisor	11
2.2	Ejemplo de aplicación del backfilling	13
3.1	Vínculo entre roles de SCRUM	16
3.2	Muestra de ciclo de vida de SCRUM	17
3.3	Diagrama de Gantt con la estimación temporal del proyecto	20
3.4	Estructura básica de SLURM	25
4.1	Fragmento del histórico de solicitudes de GPU	32
4.2	Esquema topológico y conexiones entre nodos	33
4.3	Sincronización de cliente NTP	34
4.4	Resultado tras probar munge	35
4.5	Resultado del montaje de NFS	36
4.6	Ejemplo de salida del comando slurmd -C	39
4.7	Lista de controladores de NVIDIA disponibles	41
4.8	Salida del comando nvidia-smi	41
4.9	Demostración de la política de backfill	44
4.10	Prueba de cgroups sobre núcleos de CPU	45
4.11	Salida de la prueba de restricción de recursos a nivel de cuenta	46
4.12	Fragmento de la tabla de asociaciones recursos-cuenta en SLURM	46
4.13	Agregación de fuente de datos a Grafana	47
4.14	Asignación de núcleos y temperatura de procesadores en partición de cómputo	48
4.15	Disponibilidad y asignación de memoria RAM en partición de cómputo	48
4.16	Exposición de metadatos sobre ejecuciones procesadas	48
4.17	Visualización del estado de las GPUs en partición de cómputo	48
5.3	Panel principal de Hyper-V	50
5.1	Configuración estática del adaptador de red en Windows	51

5.2	Instalación del rol de Hyper-V	51
5.4	Creación del vSwitch en Hyper-V	52
5.5	Activación de protocolo ICMP	52
5.6	Asistente de máquina virtual en Hyper-V	53
5.7	Configuración de postfix	54
5.8	Pantalla de login de Proxmox	55
5.9	Desactivación de interfaz de red física en Proxmox	55
5.10	Creación de adaptador puente en Proxmox	56
5.11	Subida de una imagen ISO a Proxmox	57
5.12	Asistente de máquina virtual en Proxmox	57
5.13	Montaje de NFS en ESXi	58
5.14	Topología del grupo de puertos en ESXi	59
5.15	Asistente de máquina virtual en ESXi	59
5.16	Activación de passthrough sobre las tarjetas gráficas	60
5.17	Designación de dispositivos PCI	61
6.1	Matriz de correlaciones entre métricas principales	65
6.2	Promedio del número de eventos en CPU	66
6.3	Evolución del ancho de banda en memoria por iteración	66
6.4	Correspondencia entre eventos de CPU y su latencia promedio	67
6.5	Correspondencia entre operaciones I/O de lectura y su latencia promedio	67
6.6	Correspondencia entre operaciones I/O de escritura y su latencia promedio	67
6.7	Evolución temporal de la tasa de bits en red	68
6.8	Promedio del número de retransmisiones de paquetes por intervalo	68
6.9	Diagrama de caja sobre la variabilidad de la ventana de congestión	69
6.10	Tiempos de ejecución del modelo de Tensorflow por iteración	69
6.11	Precisión promedio por etapa de las predicciones del modelo de Tensorflow para SLURM y ESXi	72
6.12	Penalización promedio por etapa sobre las predicciones de Tensorflow para SLURM y ESXi	72
6.13	Medición progresiva con gpu-burn del rendimiento en FLOPS de las GPU en SLURM y ESXi	72
6.14	Evolución de la temperatura de las GPU en SLURM y ESXi durante ejecución de gpu-burn	72

Índice de tablas

3.1	Distribución de horas dedicadas	22
3.2	Desglose de costes	22
3.3	Clasificación de los riesgos	24

Capítulo 1

Introducción

El desarrollo informático experimentado en los últimos años ha favorecido el auge de la simulación científica y la Computación de Altas Prestaciones (HPC), en especial con la popularización de la inteligencia artificial y los modelos de aprendizaje profundo. No obstante, esto también ha traído consigo no sólo un aumento en la heterogeneidad y volumen de las fuentes de datos, sino también una mayor complejidad en su procesamiento. En consecuencia, las tarjetas gráficas o GPU se han vuelto un recurso indispensable para agilizar los proyectos de este tipo, llegando incluso a dominar una gran parte del mercado tecnológico [1].

Asimismo, desde la pandemia se ha disparado la cantidad de servicios digitalizados, fomentando la necesidad de infraestructuras escalables y potentes capaces de adaptarse a cargas de trabajo variables. Aquí es donde entran los Centros de Procesamiento de datos (CPD), que gracias al uso de las tecnologías de virtualización y la aparición de la nube se han convertido en un pilar esencial en este ámbito, dando una mayor accesibilidad a los recursos de cómputo y facilitando tanto la agregación de sistemas de diversa índole como el despliegue de entornos portables y aislados. Tal es su relevancia que muchos de los aplicativos web que conocemos hoy en día (AWS, Azure, Google...) se sustentan sobre estas instalaciones [2].

En definitiva, hemos pasado de arquitecturas locales y monolíticas a una etapa marcada por la interoperatividad y la rapidez con la que se comuniquen resultados concluyentes sobre las peticiones. Pese a ello, este avance también plantea ciertos desafíos, sobre todo en lo relativo a la dependencia de soluciones propietarias y las condiciones impuestas por los proveedores de dichas plataformas. Dentro de este contexto, el área de sistemas del Centro de Investigación en Tecnologías de Información y Comunicación (CITIC) ha sufrido recientemente una serie de cambios en cuanto a los costes y políticas de licenciamiento sobre la gestión de recursos que utilizan GPUs, generando un escenario de incertidumbre en el que parte del *hardware* alojado corre el riesgo de quedar inoperativo en un futuro próximo.

Por tanto, la organización se ve en la necesidad de explorar diferentes soluciones para alojar todos los elementos críticos a nivel computacional, a fin de evitar que los equipos de

investigación y los administradores de sistemas vean comprometidos sus esfuerzos.

1.1 Objetivos

Teniendo en cuenta el escenario planteado, el desarrollo de este proyecto se puede esquematizar en dos puntos clave:

- Desplegar una propuesta para gestionar cargas de trabajo intensivas y solucionar la problemática mencionada con anterioridad. La idea es diseñar un **sistema de colas** con el que facilitar la delegación de las tareas de cómputo en diferentes nodos, escogiendo aquella opción con un nivel de rendimiento aceptable y que mejor se adapte a los requisitos del centro. En consecuencia, es necesario comprender el funcionamiento de la infraestructura actual y configurar un entorno de pruebas adecuado para la administración del sistema.
- Establecer un **análisis comparativo** de diferentes plataformas de virtualización para determinar si es viable una migración de los componentes menos críticos. Esto implica seleccionar varias alternativas de entre todo el ecosistema de hipervisores, evaluar tanto su capacidad técnica como las principales ofertas y funcionalidades, y contrastarlas con las especificaciones de la arquitectura mantenida por la organización. De esta forma, tomando como referencia dicho análisis, se busca reflexionar sobre la conveniencia de conservar los elementos existentes, así como documentar un plan de migración que minimice el tiempo de inactividad durante el traslado de los servicios.

Así pues, se han considerado los siguientes puntos de cara a garantizar el éxito del proyecto:

- Familiarizarse con los centros de datos, componentes e interacciones principales, además de establecer una correlación entre las cargas de trabajo y la demanda de recursos de cómputo presente en el **CPD** de la entidad.
- En cuanto al sistema de colas, se espera adquirir conocimientos básicos acerca de los sistemas de gestión y las tecnologías relacionadas con este ámbito, tratando de clarificar aspectos como el tipo de colas a implementar, su capacidad y/o el método utilizado para priorizar las tareas. Por otro lado, se deben configurar las condiciones de distribución de recursos y cargas de trabajo, con las que poder simular diferentes escenarios de gestión.
- Con respecto a los hipervisores, se pretende exponer un estudio de las soluciones escogidas (costes, rendimiento, restricciones de uso, etc.) y el procedimiento de despliegue de las mismas.

- Reconocer zonas críticas en la disposición actual del centro de datos, para asegurar una integración correcta en caso de migrar la arquitectura.

1.2 Estructura de la memoria

Una vez contextualizada la propuesta y los objetivos a alcanzar, en lo que resta del documento detallaremos tanto la metodología seguida durante el desarrollo como las fases en las que se ha dividido el proyecto, para posteriormente profundizar sobre las tareas realizadas en cada una. Finalmente, se expondrá un apartado con los resultados obtenidos y las conclusiones principales que se han extraído, además de una serie de líneas futuras que permitan expandir dicho trabajo.

Teniendo esto en mente, se ha distribuido el contenido de la siguiente forma:

Capítulo 1 Introducción: Contextualización de la propuesta y su finalidad.

Capítulo 2 Conceptos previos: Descripción de los elementos más destacados que conforman el desarrollo del proyecto.

Capítulo 3 Metodología y planificación: Explicación del ciclo de desarrollo empleado, su planificación y las tecnologías utilizadas para su compleción.

Capítulo 4 Despliegue del clúster: Exposición detallada de la infraestructura a desplegar, su puesta en marcha y la simulación de cargas de trabajo.

Capítulo 5 Configuración de virtualizadores: Ídem al caso anterior, haciendo referencia a cada una de las plataformas alternativas y a los pasos seguidos para configurar sendos entornos.

Capítulo 6 Resultados obtenidos: Mecanismos para la evaluación del rendimiento tanto del clúster como de los virtualizadores con respecto a la arquitectura original.

Capítulo 7 Conclusiones y trabajo futuro: Análisis del éxito del proyecto según los resultados previos y qué lecciones se han aprendido.

Capítulo 2

Conceptos previos

A la hora de profundizar en el desarrollo de un proyecto debemos contar con una visión adecuada sobre su entorno y cómo este ha evolucionado. Este capítulo está pensado para dar una noción de los elementos más relevantes, visualizando la relación que existe entre estos y los objetivos de negocio.

2.1 Qué es HPC

La [Computación de Altas Prestaciones](#) es un campo que abarca una gran cantidad de facetas tecnológicas [3], cuyo objetivo es maximizar la capacidad de procesamiento computacional de un sistema mediante recursos con especificaciones que sobrepasan las de cualquier equipo informático estándar [4]. En ocasiones es tratado como un concepto atemporal y de gran variabilidad, motivado por la constante aparición de nuevas técnicas y el crecimiento exponencial de la magnitud de muchos proyectos científicos [5]. Por ende lo que se percibía décadas atrás como un "supercomputador" puede ser en la actualidad algo totalmente distinto.

Asimismo, es considerado una parte fundamental en el desarrollo de la ciencia [3, 6, 7], con aplicaciones en multitud de áreas como la medicina o la astrofísica para la simulación de escenarios de alto calibre.

2.1.1 Modelos de computación

La distribución de los componentes y cómo se gestionan es un aspecto de gran importancia si se quiere aprovechar el despliegue técnico de uno de estos sistemas. Por este motivo, es habitual clasificarlos en tres grandes grupos [8, 9]:

- **De clúster:** Consiste en la interconexión de elementos redundantes para formar un único sistema capaz de ofrecer tanto un cómputo más eficiente como alta disponibilidad, tolerancia a fallos y balanceo de carga. Son bastante útiles en tareas dedicadas a la

minería y procesado masivo de datos. Sin embargo, esta disposición centralizada también presenta inconvenientes en cuanto a su capacidad para adaptarse a cambios en la demanda.

- **De cuadrícula o grid:** Se puede entender como una asociación de recursos ubicados en diferentes dominios administrativos para la resolución de un objetivo común. Si bien permite una mayor heterogeneidad y desacoplamiento entre sus componentes, también implica una monitorización más exhaustiva de las interacciones entre dominios. Normalmente se emplean en actividades de modelado de características que requieren un alto nivel de precisión como la meteorología o la física nuclear.
- **En la nube:** Hace referencia al conjunto de componentes físicos y virtuales que habilitan el despliegue de servicios a través de Internet. A diferencia de los modelos tradicionales, su principal característica reside en el aprovisionamiento transparente y dinámico de recursos, así como en su habilidad para escalar de forma flexible conforme se modifican los requisitos de las aplicaciones.

2.1.2 Tendencias actuales

Desde sus inicios, la disciplina de HPC ha experimentado un progreso constante en cuanto a complejidad, portabilidad de servicios y sobre todo capacidad de procesamiento, siendo un referente para exponer aquellas tecnologías con mayor probabilidad de popularizar el mercado durante los años venideros [7]. Además, la creación de proyectos como el *Top500* permitió promover y aportar rigor a este fenómeno, categorizando a los diferentes “supercomputadores” a través de una serie de *benchmark*.

Esta década en particular ha supuesto una transformación radical del panorama, abogando por una necesidad de especializarse y mejorar el funcionamiento de las infraestructuras [10], no solo a nivel técnico (e.g. superar la barrera del exaescalado) sino también en lo que respecta a la eficiencia energética de los componentes, llegando incluso a diseñar un *ranking* propio para esto último [11].

En las siguientes secciones mostraremos algunos de los ámbitos que han marcado de forma notoria esta transición.

Big Data e IA

El rápido incremento en volumen y variabilidad que han experimentado la mayoría de fuentes de información ha provocado que el análisis y extracción de características se vuelva en muchos casos una tarea bastante costosa. Con una fuerte presencia del contenido audiovisual, y debido a la ausencia de un formato estándar para su representación, se vuelve indispensable adaptar los métodos tradicionales para que puedan reconocer patrones relevantes

y garanticen unos umbrales de calidad. La inteligencia artificial ha permitido cimentar este camino mediante el uso de tarjetas gráficas para el entrenamiento de modelos de aprendizaje automático [12], lo cual también se ha reflejado en la reciente colaboración de diversas compañías tecnológicas [7] con empresas dedicadas a estas actividades (e.g. Microsoft con OpenAI). De esta forma se ha logrado una convergencia entre las diferentes ramas, incorporando estos métodos de manejo e inferencia de datos como un mecanismo de retroalimentación en el desarrollo de proyectos más sofisticados [13].

Optimización energética

Como se mencionó al comienzo de esta sección, existe un gran interés en este sector industrial por minimizar los gastos energéticos, puesto que albergar sistemas computacionales cada vez más potentes exige una demanda acorde de electricidad, hasta el punto de convertirse en los últimos años en una situación desorbitada [14]. La computación en la nube supuso el primer paso hacia esta economización, al provocar que una gran cantidad de empresas migrasen su arquitectura física a una disposición basada en microservicios virtualizados [7], reduciendo de forma considerable el número de estructuras a mantener. En lugar de aportar más y más recursos, se busca refinar la gestión de los ya existentes y asegurar que estén disponibles en todo momento, así como minimizar la tasa de fallos en las interacciones críticas [15, 16]; al fin y al cabo un error puede desembocar en largos períodos de inactividad, y cuanto más se extiendan mayor es la pérdida asociada.

En [11] los autores reflexionan acerca de estos problemas, contrastando de forma gráfica el nivel de impacto de los computadores del *Top500* y cómo ha evolucionado la relación entre sus emisiones y el rendimiento potencial.

Aceleradores y TPUs (*Tensor Processing Units*)

Dado que todos los esfuerzos se están invirtiendo en optimizar aquello relativo a las simulaciones, era de esperar que se contemplase el diseño de elementos externos para obtener tiempos de ejecución más ajustados. La aceleración basada en *hardware* es la opción más señalada, en especial por el creciente uso de las **GPU** y su extrapolación para el procesamiento paralelo de diversos cálculos [13]. Es por ello que las primeras versiones de aceleradores se asemejan más a grupos de tarjetas con conexiones PCIe. Sin embargo, en [10] también creen que estos agregados se acabarán convirtiendo en sistemas con recursos dedicados y una monitorización ajena a la del resto de la infraestructura. Por otro lado, la expansión del aprendizaje profundo ha permitido desarrollar unidades específicas denominadas TPU, una serie de motores matriciales con los que agilizar el entrenamiento de modelos generativos y redes neuronales.

Si bien es cierto que los aceleradores gráficos son los más habituales, también se pueden aplicar a otros paradigmas y componentes, pero todos comparten el mismo objetivo: simplificar los ciclos de acceso a la memoria dinámica.

2.2 Centros de datos

Se trata de instalaciones que agrupan por medio de redes de comunicación una serie de recursos informáticos [17], cuya función principal es permitir el alojamiento centralizado de datos y aplicaciones. Por tanto, para que una infraestructura de este tipo sea mínimamente funcional es necesario que incluya componentes que ofrezcan potencia de computación, dispositivos de almacenamiento y equipamiento de red para exponer dichos sistemas a Internet o a otros departamentos dentro de una misma organización [18]. Todos estos recursos se suelen distribuir en varios armarios con chasis modulares denominados *racks* [2], los cuales gracias a la elevada densidad en su disposición pueden llegar a contener los medios necesarios para suministrar alimentación eléctrica, mecanismos de refrigeración e incluso sensores de diversos tipos.

2.2.1 Evolución

Originalmente, la arquitectura que residía en estas ubicaciones estaba formada por computadoras centrales (*mainframes*) y múltiples dispositivos electromecánicos [19], siendo el ENIAC la primera referencia que se tiene de un CPD al uso [4, 18, 20]. No obstante, este paradigma cambió con la aparición de los microprocesadores y la transición de memorias magnéticas a semiconductores [20], dando lugar a formaciones monolíticas de servidores con discos locales que, si bien presentaban un desaprovechamiento considerable tanto de CPU como de espacio físico y almacenamiento, consiguieron prevalecer durante varias décadas.

No obstante, hacia los años 80 comenzó la masificación de los ordenadores personales; surge una necesidad por parte de las empresas de heterogeneizar su *software* y migrar de los “silos” de servidores a una localización común para todos los componentes. Esta transición fue facilitada sobre todo por los avances realizados en el campo de la virtualización, especialmente en lo que respecta a la migración en vivo de componentes y máquinas [19].

Pese a ello, no sería hasta los años 2000 cuando con la tendencia de la computación en la nube y la creciente demanda en el despliegue de páginas web [20] se verían los frutos de dicha inversión, convirtiéndose indirectamente en una solución para consolidar¹ la cantidad de servidores físicos en las salas (proceso *Physical to Virtual* o *P2V*) y reducir de manera significativa la cantidad de energía consumida y su huella de carbono [19, 21]. También surgieron

¹ El ratio de consolidación determina el nivel de reducción obtenido en una escala cualquiera. En este caso, consiste en alojar dentro de una máquina física tantas máquinas virtuales como sean posibles a fin de alcanzar un consumo eficiente y un mejor aprovechamiento de sus capacidades.

en esta etapa los predecesores de las herramientas de VMWare y Microsoft dentro de este ámbito, las cuales son de las más utilizadas hoy en día y de las que hablaremos en profundidad más adelante.

2.2.2 Tipos de infraestructura

En la actualidad resulta habitual categorizar a los CPD atendiendo a su grado de disponibilidad, es decir a la cantidad de tiempo que el servicio se encuentra inactivo, lo que a su vez depende en gran medida del esquema que se utilice en la distribución del equipamiento TI y en la presencia de elementos redundantes. El organismo [Telecommunications Industry Association \(TIA\)](#) emite una serie de certificaciones para poder verificar estos requisitos, entre ellos el estándar ANSI/TIA, distinguiendo así entre cuatro clases diferentes [18, 22]:

- **Tier I (28.8 horas de inactividad al año):** Es el nivel más básico, en el que no hay ningún atisbo de redundancia, y puede disponer o no de algún SAI o generador de corriente. Es ideal para negocios de pequeña magnitud.
- **Tier II (22 horas/año):** Presenta redundancia únicamente en los componentes, pero los canales de comunicación siguen siendo individuales.
- **Tier III (1.6 horas/año):** Tanto canales como equipamiento son redundantes, sumado a la capacidad de realizar el mantenimiento de alguno de los componentes sin que ello afecte al rendimiento del resto o que la infraestructura se quede fuera de servicio.
- **Tier IV (0.4 horas/año):** Es bastante similar al nivel anterior, pero es capaz de tolerar un fallo en cualquier sector del sistema y que este no se encuentre caído. Es propio de organizaciones con operaciones financieras a gran escala, como es el caso de Inditex.

2.2.3 Acerca de la nube

Como hemos visto anteriormente, los centros de datos han transicionado hacia un enfoque basado en salas amplias a cargo de una única entidad, gracias a la reducción de los costes que supone el minimizar el número de servidores infrautilizados [2, 21]. No obstante, también conlleva que esa sensación de propiedad en la utilización de los recursos se vuelva más abstracta y confusa, provocando que las organizaciones de TI aspiren a desarrollar modelos de suscripción dependientes de la demanda. Aquí es donde entra en juego el concepto de la nube, un entorno que ofrece aprovisionamiento de forma rápida, flexible y ante todo escalable (ya sea público, privado o incluso una solución híbrida) [19], además de descargar ciertas responsabilidades administrativas en la parte de los proveedores.

Se puede decir entonces que la nube es un término que se sustenta y gira en torno a la virtualización, pero con una finalidad totalmente distinta [23] ya que, en vez de optimizar la

capacidad de cómputo de una infraestructura concreta, ofrece un método de acceso global y de bajo coste a sus recursos o a una parte de estos en un instante determinado.

2.3 Virtualización: técnicas y plataformas

Hasta ahora se ha hablado del trayecto que ha desembocado en la creación de los servidores virtualizados, pero es igual de importante comprender los métodos que se utilizan para llevarlo a la realidad y qué tecnologías son representativas de cada grupo.

A grandes rasgos, la virtualización consiste en generar una visión lógica de dispositivos físicos para que múltiples servicios puedan utilizar los mismos recursos y compartirlos, con independencia de su origen [24]. Esto otorga al usuario final una capa que oculta la complejidad del *hardware* residente, sin que sea consciente de las tareas de gestión y de designación del mismo. Lo más habitual es aplicarlo sobre un sistema operativo, aunque también existen varias opciones para almacenamiento, red y a nivel de aplicativos de cliente [23, 25].

En lo que respecta a servidores, el objetivo es encapsular varios sistemas dentro de un equipo anfitrión, el cual actúa como un intermediario entre estos y los componentes para resolver sus peticiones sin que entren en conflicto ni haya problemas de compatibilidad ante un cambio en la arquitectura [26]. Este elemento central puede ser considerado de varias maneras dependiendo de cómo se realice la abstracción. Principalmente, nos encontramos con tres enfoques [25]:

- **Basado en hipervisor:** Se presenta en forma de *software* con el rol descrito previamente. A su vez se divide en dos clases (Figura 2.1):
 - **Tipo 1 o bare-metal:** El hipervisor está ubicado directamente sobre los componentes físicos. Servicios como ESXi de VMWare o Hyper-V de Microsoft forman parte de esta categoría.
 - **Tipo 2:** El *host* físico contiene su propio sistema operativo, y el hipervisor depende de este para gestionar los componentes y satisfacer las solicitudes de las máquinas virtuales. VirtualBox y VMWare Workstation son ejemplos claros de este grupo.
- **Emulación:** Traduce el conjunto de instrucciones de una arquitectura para replicar su comportamiento en otro sistema. Suele utilizarse para ejecutar programas que se hayan descontinuado o que debido a las especificaciones de su diseño no son compatibles en entornos modernos.
- **Basado en contenedores:** Las máquinas virtuales se consideran procesos del sistema anfitrión que comparten sus recursos y son gestionados de forma centralizada mediante un elemento controlador.

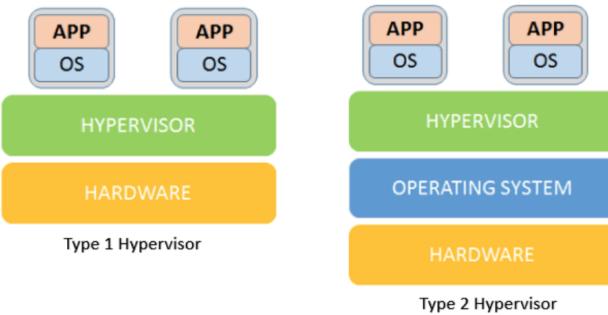


Figura 2.1: Visualización de tipos de hipervisor

2.4 Sistemas de gestión de recursos

Los [Resource Management Systems \(RMS\)](#) comprenden un gran conjunto de operaciones destinadas a la planificación de cargas de trabajo y al aprovisionamiento de los componentes necesarios para efectuarlas [27]. Estas actividades resultan esenciales en un centro de datos, puesto que debe ser capaz de seleccionar los elementos que mejor se ajusten a las necesidades de cada solicitud, de modo que estas sean procesadas en un tiempo razonable y no comprometan en gran medida el rendimiento general de la infraestructura.

Un sistema de este tipo se divide en dos elementos [28]: el *resource manager*, que se comunica con los nodos del clúster para enviarles las tareas a realizar; y el *scheduler*, que evalúa la disponibilidad de los recursos y establece qué porción de los mismos le corresponde a cada trabajo, así como el orden en el que se llevan a cabo según las políticas del entorno [14]. A su vez los planificadores pueden ser estáticos o dinámicos [29], en función de si conocen de antemano los parámetros de ejecución (memoria, número de CPUs, restricciones de tiempo, etc.) y de su capacidad para manipular los recursos designados.

Por otro lado, la estructura de los RMS también se puede descomponer en las siguientes funcionalidades [30]:

- El gestor del **ciclo de vida** de trabajos, quien dispone las tareas en colas, procesa los requisitos incluidos y las ordena.
- El **gestor de recursos**, que analiza el estado de los nodos y se lo traspasa al planificador, además de recolectar información sobre las ejecuciones en curso.
- El **planificador**, encargado de asignar las peticiones a los nodos correspondientes en caso de que se encuentren disponibles.

- El **ejecutor** de trabajos, responsable de lanzarlos y manejar su posterior cierre, registrando cualquier estadística resultante.

Relacionado con el punto anterior, es importante reconocer la clase de trabajos a incorporar en la cola, teniendo en cuenta no solo si son secuenciales o paralelos sino también las características que requieren para su ejecución. De esta forma se pueden agrupar en cuatro categorías: **rígidos, evolutivos, moldeables y maleables** [14, 28]. La diferencia reside en cómo se establece el reparto de los componentes y el grado de influencia en ellos. En los trabajos rígidos es el propio usuario quien, antes de que se procese la solicitud, aporta la información correspondiente mediante una serie de indicaciones inmutables, mientras que en los evolutivos se puede modificar esta reserva en plena ejecución. Asimismo, en los moldeables el planificador es quien decide la configuración de las tareas una vez hayan sido solicitadas por el usuario, y en los trabajos maleables se da total libertad para expandir o reducir dicha asignación.

En estos entornos lo habitual es encontrarse con tareas rígidas [14], dado que aquellas con una distribución dinámica (i.e. moldeables y maleables) requieren de una monitorización más intensiva y son más complejas de manejar, derivando eventualmente en una sobrecarga de los servidores de cómputo.

2.4.1 Estrategias de programación

Cuando se carece en un instante determinado de recursos suficientes para satisfacer un nuevo trabajo, el planificador lo mantiene en una cola de espera hasta que se produzca una liberación por parte de otra tarea. La forma en la que se organicen en esta cola (y que por ende determina su orden de procesamiento) dependerá sobre todo de las heurísticas consideradas, tales como la cantidad de recursos solicitados o el tiempo estimado de ejecución [14]. Los algoritmos más populares suelen ser aquellos con una implementación más simple, de entre los que destacan: FCFS, donde los primeros que entran son los primeros en salir; y SJF/LJF, que procesa primero aquellos trabajos más cortos o más largos, según la variante utilizada [31].

No obstante, en cualquiera de estas aproximaciones puede ocurrir que si la tarea ubicada al principio de la cola es incapaz de obtener los requisitos necesarios, pese a haber algún elemento inutilizado, entonces todas las que vienen consecutivamente permanecerían bloqueadas lastrando la eficiencia del sistema. En consecuencia aparecieron estrategias como el *backfilling* para mitigar esta situación. Esta técnica [14, 29, 31, 32] consiste en revisar todos los trabajos que se encuentran en la cola en un determinado momento y escoger aquel que mejor se adapte a la disponibilidad actual (Figura 2.2).

Con ello se logra priorizar indirectamente a aquellas tareas de poco tamaño y que tienden a completarse en poco tiempo, agilizando el vaciado de la cola y garantizando un flujo cons-

tante de las peticiones. Esta medida puede aplicarse en modo simple o conservador [14, 32], dependiendo de cómo se estime el retraso causado por las actividades adelantadas.



Figura 2.2: Ejemplo de aplicación del backfilling

2.4.2 Relación con HPC

Resulta sencillo concluir que los planificadores de recursos constituyen un aspecto esencial en los sistemas de cómputo [8]; no es viable buscar un alto grado de productividad de los componentes sin contar con un esquema claro sobre cómo se deben distribuir o qué características se deben priorizar ante la llegada de nuevos trabajos. Esta situación se manifiesta independientemente de la época considerada y los tipos de tareas a procesar en cada entorno, dado que en la actualidad sigue siendo una opción utilizar aplicativos que surgieron a principios de siglo o variantes de los mismos, lo cual además enfatiza su robustez ante los cambios drásticos de las tecnologías involucradas [30]. Por tanto, supervisar su rendimiento en busca de cualquier fallo se torna en una pieza clave en el funcionamiento del gestor, ya que en caso contrario no se podrá sacar el máximo partido a las inversiones realizadas.

2.5 Benchmarking

Este concepto engloba todas aquellas técnicas que permiten llevar a cabo una evaluación continua y competitiva acerca de un producto o servicio respecto a lo ya conocido [33]. En otras palabras, es un proceso que tiene como objetivo medir la calidad de un elemento utilizando otros de su mismo entorno como ejemplos a seguir [34]. En el mundo de la computación es habitual clasificarlos en dos grupos: sintéticos y basados en aplicaciones [35]. Los primeros pretenden probar el funcionamiento de una parte de los componentes, utilizando simulaciones controladas y reproducibles, mientras que los del segundo tipo recrean cargas de trabajo reales para monitorizar el comportamiento general del sistema, así como determinar su rendimiento en distintas situaciones.

Para que un programa pueda ser considerado como un *benchmark*, su punto fuerte debe estar en alguno de los siguientes aspectos [35]:

- **Relevancia:** Se traduce en cómo de cerca está el programa de satisfacer los intereses del usuario y lo afín que sea al contexto de las áreas consideradas. Es importante que alcance un compromiso entre escalabilidad y especificidad.
- **Replicabilidad:** Es capaz de proporcionar resultados lo más similares posible de forma consistente, independientemente del sistema a utilizar y de la condición en la que este se encuentre.
- **Balance:** Debe establecer reglas y/o restricciones que permitan realizar ejecuciones próximas a un caso real.
- **Verificabilidad:** Si incluye algún método para validar el rigor de los resultados obtenidos, o bien muestra al usuario ciertos detalles que se han tenido en cuenta para configurar la simulación.
- **Usabilidad:** Cómo de amigable o intuitivo es el programa de cara a asistir al usuario en la parametrización de las ejecuciones.

En este trabajo se ha puesto el foco en las comparativas sintéticas, al ser las que mejor permiten estimar la capacidad técnica de los servidores y las que aportan una visión más reveladora del contraste entre los valores base y aquellos que se obtienen como resultado de considerar por medio un modelo de virtualización. No obstante, también se han diseñado varios códigos que servirán como referentes de los *test* de aplicaciones, adaptándolos a casos de uso plausibles en un ambiente de estas características.

Capítulo 3

Metodología y planificación

EN todo proyecto es importante tener claro cuál va a ser la metodología a utilizar, ya que de no aplicarla correctamente puede condicionar su desarrollo y por ende el éxito del mismo. En este capítulo se justificará aquella que ha sido seleccionada junto a sus atributos más destacados. También se expondrá el plan elaborado en base a dicha estrategia, incluyendo una previsión del tiempo a dedicar y los costes del desarrollo junto a sus principales limitaciones, para luego describir brevemente cada uno de los artefactos *software* involucrados en el proceso.

3.1 Metodología

Para este escenario se ha decidido recurrir a **SCRUM**. Se trata de un marco de trabajo ágil e incremental, por lo que está basado en la división del desarrollo de un producto en diferentes versiones que van escalando a nivel de funcionalidades [36]. Surgió en los años 80 como una opción divergente a las estrategias tradicionales (e.g. método en cascada), dando la opción de evitar ciertos pasos o procedimientos que pudiesen entorpecer el avance del desarrollo. De esta forma, se logra fomentar una mentalidad centrada en el rendimiento, con una comunicación simple y efectiva de los requisitos.

La idea que persigue es fragmentar el ciclo de vida de un proyecto en una secuencia de iteraciones de corta duración llamadas *sprints*, las cuales suelen extenderse de 1 a 3 semanas e involucran a equipos de tamaño reducido. Esto asegura una visión unificada del concepto a implementar en cada etapa, además de permitir una mayor sensación de tangibilidad en el resultado de las mismas y que se realice una mejor estimación de las tareas contenidas [37].

Por otro lado, esta metodología consta de varios elementos clave: roles, artefactos y ceremonias.

3.1.1 Roles

Los roles garantizan la definición de una jerarquía estable e intuitiva, en la que todos los miembros presentan una relevancia uniforme y responsabilidades concretas (ver figura 3.1). Dentro de estos tenemos:

- **Product owner:** Es quien determina el enfoque del producto a implementar, por lo que se encarga principalmente de la extracción y consolidación de los requisitos, así como de resolver los conflictos entre el equipo y los clientes u otros involucrados. No se debe confundir con un gestor de proyectos, ya que va más allá de este perfil al ser una referencia para la validación del desarrollo y el individuo que toma las decisiones vitales de negocio.
- **Scrum Master:** Este rol gestiona cualquier problema que pudiera ocasionar un retraso durante el ciclo de vida, supervisa el desempeño del equipo y verifica que se está cumpliendo con las pautas dictaminadas por la metodología. De esta forma actúa como un consejero (*coach*) que guía a los miembros hacia el camino correcto, tratando de maximizar la productividad y asegurándose de que dispongan de los recursos necesarios para lograrlo.
- **Equipo de desarrollo:** Son responsables de que el producto final alcance unos estándares de calidad aceptables. A diferencia de las metodologías tradicionales, se muestra como una entidad autodidacta en lo que respecta a la priorización de los requisitos abordables en un *sprint*, la construcción y delegación de las tareas representativas y en la retroalimentación activa sobre la precisión en las estimaciones del tiempo de trabajo.

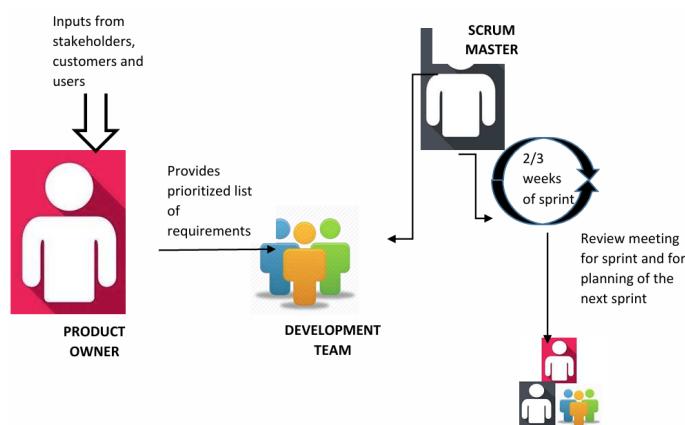


Figura 3.1: Vínculo entre roles de SCRUM

3.1.2 Artefactos

Constituyen toda la documentación que se genera durante la extracción de requisitos y al planificar los *sprints*. En ella encontramos:

- **Product backlog:** Agrupa todos los requisitos y funcionalidades que el *product owner* ha identificado en formato de historias de usuario, junto a un valor de prioridad y un tiempo de estimación.
- **Sprint backlog:** Representa las tareas derivadas en cada iteración por el equipo de desarrollo de los requisitos del documento mencionado anteriormente.
- **Release Plan:** Describe qué funcionalidades permiten construir un producto mínimo viable, además de la fecha y el coste previstos.

Opcionalmente se pueden obtener tanto para el producto como para las iteraciones una gráfica con la evolución temporal prevista de los valores de saturación a lo largo del proyecto, también llamadas *burndown charts*. Por otra parte, en la figura 3.2 se puede apreciar la relación entre algunos de estos artefactos y las ceremonias, de las cuales hablaremos a continuación.

3.1.3 Ceremonias

Son aquellos eventos relacionados con la puesta en marcha de las iteraciones, entre los que se incluyen las reuniones de planificación, revisión y retrospección de los *sprints*. Todas ellas son relativamente cortas (entre 1 y 2 horas) y suelen ser convocadas por el *scrum master*.

También se pueden llevar a cabo reuniones diarias en las que los miembros del equipo de desarrollo exponen los avances realizados desde la última sesión, qué es lo que pretenden completar antes de la siguiente y los impedimentos que pudieran surgir entre ambas.

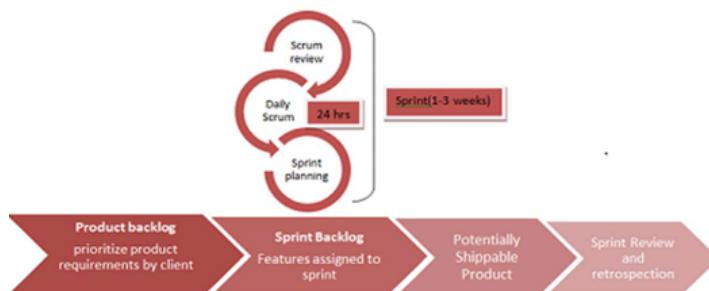


Figura 3.2: Muestra de ciclo de vida de SCRUM

3.1.4 Justificación

Este proyecto, dada su naturaleza y los objetivos que pretende alcanzar, mantiene una alta probabilidad de que las características solicitadas inicialmente vayan evolucionando conforme se avanza en el desarrollo, por lo que recurrir a una metodología tradicional provocaría más inconvenientes que beneficios. Además, la flexibilidad de SCRUM en el establecimiento de las fases a abordar y los procedimientos para llevarlas a cabo otorga a los miembros del equipo un amplio grado de libertad a la hora de escoger la dirección que va a seguir la propuesta [36], a la vez que ofrece una capacidad de respuesta más eficiente ante la aparición de cambios imprevistos.

3.2 Planificación

Tal y como se ha mencionado en el capítulo introductorio, este proyecto presenta dos grandes bloques: el sistema de colas y la implementación de diferentes virtualizadores para contrastar su rendimiento con el de la infraestructura actual. Por tanto, al aplicar la metodología SCRUM no resulta descabellado considerar a cada una de las propuestas como un desarrollo aparte con sus propias iteraciones.

3.2.1 Definición del backlog

En este proyecto los principales interesados son los administradores de los sistemas y todo aquel que ocupe un puesto de investigador en la entidad, por lo que a continuación se expondrán los puntos de vista de dichos perfiles en relación con los productos a implementar.

Investigadores

Como investigador, quiero que el **sistema de colas**...

- Permita enviar peticiones de ejecución y sus parámetros de forma sencilla.
- Garantice la seguridad e integridad de los datos y resultados obtenidos.
- Ofrezca una visión rápida y ágil del estado de las tareas y su progreso.
- Sea intuitivo, eficiente y de fácil aprendizaje, para ralentizar lo menos posible mis avances.
- Disponga de un soporte con tiempos de respuesta no muy prolongados ante la aparición de fallos.

Y quiero que el **virtualizador**...

- Garantice la modificación dinámica de los recursos de las máquinas virtuales (CPU, memoria RAM, almacenamiento, etc.).
- Permita migrar mis máquinas, o en su defecto generarlas a partir de plantillas, para no perder el trabajo previo.

Administradores de sistemas

Como administrador, quiero que el **sistema de colas**...

- Permita configurar de forma flexible los nodos y las reglas de asignación.
- Disponga de algún mecanismo de monitorización del consumo de recursos.
- Defina permisos de acceso, para restringir las redes o usuarios que puedan utilizarlo.

Y quiero que el **virtualizador**...

- Conserve las funcionalidades más importantes de la infraestructura actual.
- Ofrezca una mayor continuidad y autosuficiencia de los servicios, evitando depender de factores externos.

3.2.2 Estimación temporal

Tras haber clarificado los requisitos principales, es necesario extraer los casos de uso y determinar las fases en las que se va a dividir el proyecto, junto con la previsión de horas a dedicar en cada tarea (tabla 3.1). Gracias a ello podemos obtener el [diagrama de Gantt](#) de la figura 3.3, que a su vez nos ayudará a establecer los costes del proyecto y los riesgos potenciales de cada fase (apartados 3.2.3 y 3.2.5 respectivamente).

Iteración 1 *Diseño del sistema de colas.*

Tarea 1 Análisis del escenario actual y planteamiento de la nueva arquitectura.

Iteración 2 *Composición del clúster.*

Tarea 1 Configuración e interconexión de los nodos.

Tarea 2 Gestión y autenticación de usuarios.

Iteración 3 *Puesta en marcha del scheduler.*

Tarea 1 Creación del almacenamiento compartido.

Tarea 2 Instalación y designación de roles en el clúster.

Tarea 3 Configuración de reglas y políticas de encolado.

Tarea 4 Generación de la base de datos para el *accounting*.

Iteración 4 *Evaluación del rendimiento.*

Tarea 1 Despliegue y simulación de cargas de trabajo.

Tarea 2 Análisis de resultados obtenidos.

Iteración 5 *Refinamiento y monitorización de la infraestructura.*

Tarea 1 Bastionado de las interacciones con los servidores.

Tarea 2 Optimización y ajuste del procesamiento de peticiones.

Iteración 6 *Diseño y despliegue de los hipervisores.*

Iteración 7 *Documentación y análisis comparativo de las soluciones.*

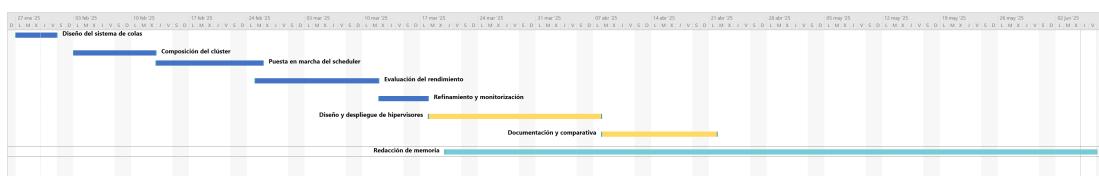


Figura 3.3: Diagrama de Gantt con la estimación temporal del proyecto

Cabe señalar que previo a estas tareas se llevó a cabo una labor de investigación para familiarizarse con las tecnologías involucradas y cómo se conforma la estructura de un gestor de recursos, lo cual se completó en torno a 3 semanas. Asimismo, para el cálculo de las horas totales se asumió una dedicación semanal por parte del alumno de alrededor de 30 horas, y que las reuniones convocadas se finalizaron por lo general al cabo de 1 hora.

3.2.3 Estimación de costes

Para esta parte se vuelve imprescindible hacer una distinción entre costes humanos y materiales. En el cálculo del primer grupo se tomó como referencia los valores indicados en el BOE acerca de este convenio, donde se expone que un trabajador situado en el sector de la ingeniería informática (equivalente al nivel salarial 1) presenta un total anual de 28.664,12 € [38]. Asumiendo que un mes contiene 20 días laborables y que cada jornada es de hasta 8 horas, la cifra se traduce pues en un coste de 14,92 €/hora. Por otro lado, el salario base de un *scrum master*, que es el rol designado a los directores del trabajo, es una medida de gran variabilidad, ya que depende sobre todo de la empresa y el tipo de proyecto. Tras analizar varias fuentes se

pudo realizar una deducción del sueldo promedio, ascendiendo así a 45.000 € anuales o 23,44 €/hora. Asimismo, el número de horas en las que han estado presentes queda determinado por la cantidad de reuniones convocadas (con la duración descrita en el apartado anterior) y por la cantidad de tiempo destinado a la resolución de diferentes conflictos durante el transcurso de las iteraciones o a cualquier consulta relacionada con la revisión de esta memoria.

En el caso de los gastos materiales, se engloban aquellos recursos incluidos de forma externa para la construcción del entorno y el desempeño de las actividades del proyecto. Concretamente, el coste incurrido por los elementos físicos se ha expresado en forma de amortización por tiempo de uso a partir de la fórmula (3.1), donde $V_{residual}$ representa el precio mínimo del producto tras finalizar su vida útil (supondremos 4 años y una depreciación anual del 20%) y H_{total} la suma de horas que podría estar en funcionamiento durante dicho intervalo:

$$A_{\text{€/hora}} = \frac{C_{\text{inicial}} - V_{\text{residual}}}{H_{\text{total}}} \quad (3.1)$$

En la tabla 3.2 se hace alusión al valor total de ambos tipos de coste, desglosados en los sujetos clave extraídos a partir de la información proporcionada. Cabe resaltar que en un entorno de producción también sería de gran interés hallar los gastos indirectos (e.g. consumo eléctrico de la instalación), pero en un proyecto de este tipo determinar la exactitud de dichos valores se vuelve una tarea compleja, de modo que no se han tenido en cuenta como parte de la previsión final.

3.2.4 Seguimiento

Sobre este punto se decidió adoptar un enfoque híbrido con respecto al planteamiento original de SCRUM, aprovechando su punto fuerte de dar una mayor libertad en la gestión del calendario. De esta forma, las reuniones quedaban convocadas en intervalos de 2-3 semanas, o en su defecto al finalizar cada iteración, informando en cualquier caso a los directores acerca del progreso de las tareas conforme estas se iban completando o se experimentaba algún inconveniente.

3.2.5 Gestión de riesgos

Dentro de la planificación de un proyecto también es necesario tener en cuenta los riesgos potenciales y saber cómo mitigarlos, siendo bastante útil clasificarlos atendiendo a su influencia en el desarrollo. Así pues, hemos considerado los siguientes supuestos:

RI- 1 Problemas de compatibilidad del hardware. Puede darse el caso de que algunos de los servidores disponibles no permitan realizar ciertas configuraciones o directamente no soporten los requisitos para desplegar la solución

Iteración	Horas estimadas	Horas reales
<i>Diseño del sistema de colas</i>	30	24
<i>Composición del clúster</i>	48	60
<i>Puesta en marcha del scheduler</i>	54	66
<i>Evaluación del rendimiento</i>	66	78
<i>Refinamiento y monitorización</i>	24	27
<i>Diseño y despliegue de hipervisores</i>	90	90
<i>Documentación y comparativa</i>	42	45
Total	354	390

Tabla 3.1: Distribución de horas dedicadas

Recurso	Coste/hora (€)	Nº horas	Coste total (€)
Costes humanos	-	-	6100,08
<i>Desarrollador</i>	14,92	390	5818,80
<i>Director</i>	23,44	12	281,28
Costes materiales	-	-	61,16
<i>2x GPU QUADRO RTX A6000</i>	0,12	132	31,68
<i>2x SSD Kingston DC600M 960GB</i>	0,004	132	1,16
<i>Portátil Asus X571GT-BQ709</i>	0,08	354	28,32
Total			6161,24

Tabla 3.2: Desglose de costes

deseada, por lo que se precisaría adquirir nuevos componentes o recurrir a otras estrategias.

RI- 2 Contratiempos en la asignación de recursos al clúster. Aunque se disponga de los elementos adecuados, algunos de ellos y en especial las tarjetas gráficas pueden causar dificultades a la hora de vincularlos con servidores y máquinas virtuales.

RI- 3 Complicaciones para establecer la conectividad del entorno. Existe la posibilidad de que, tras integrar las redes y sus reglas, se encuentre algún error en el tráfico entrante/saliente que inhabilite temporalmente el acceso remoto a la infraestructura o a ciertas partes del servicio.

RI- 4 Manifestación de cortes en el suministro eléctrico. Si bien es una situación poco común en esta clase de entornos, pueden tener consecuencias ampliamente notorias en el sistema, llegando a paralizar el desarrollo por completo hasta que se recupere la alimentación.

RI- 5 Retrasos en configuración y/o bastionado de servidores. Durante la puesta a punto de los nodos puede haber algún error humano en las sentencias de los ficheros o la gestión de los servicios que provoque comportamientos inesperados durante las pruebas.

RI- 6 Retrasos en capacitación para pruebas del entorno. Además de que el sistema funcione correctamente, es importante que su uso esté bien documentado y sea comprensible para los usuarios finales. Sin embargo, es probable que en un momento dado no posean todos los conocimientos precisos y se vuelva necesario reformular las ideas transmitidas.

Id. Riesgo	Probabilidad	Impacto	Desviación esperada
<i>RI-1</i>	Media	Alto	5 días
<i>RI-2</i>	Media	Medio	3 días
<i>RI-3</i>	Media	Medio	3 días
<i>RI-4</i>	Baja	Alto	3 días
<i>RI-5</i>	Baja	Medio	2 días
<i>RI-6</i>	Alta	Medio	4 días

Tabla 3.3: Clasificación de los riesgos

En la tabla 3.3 se pueden apreciar los valores estimados¹ para cada riesgo, así como el ajuste en días que implicaría su aparición respecto a la línea base del proyecto. Este último parámetro se puede aproximar de varias maneras, pero finalmente se escogió una solución simplista a partir del *ceil* entre su probabilidad e impacto ($\lceil P * I \rceil$), dando como resultado un total de 20 días aproximadamente.

3.3 Tecnologías a utilizar

3.3.1 Desarrollo del sistema de colas

SLURM

Como bien indica su nombre, [Simple Linux Utility for Resource Management](#) [39] se presenta como una alternativa de código abierto y de fácil configuración para gestionar recursos en diferentes arquitecturas de clúster con sistemas Linux. Ofrece un *framework* con el que controlar su acceso, planificar la ejecución de las tareas y resolver cualquier conflicto en dichas asignaciones a través de colas de prioridad. Además, permite el uso de diversos *plug-in* [40] para optimizar la topología y establecer restricciones en la reserva de recursos, así como registrar las operaciones invocadas y su resultado en una base de datos, entre otros aspectos.

Generalmente, su estructura está constituida por tres grandes bloques que interactúan entre sí mediante servicios en segundo plano (figura 3.4): la parte **controladora**, en la que puede o no haber una base de datos, la parte de **cómputo** y los comandos de **usuario**.

El motivo por el que se ha seleccionado va más allá de su modularidad y accesibilidad. A

¹ Si bien se muestran en formato categórico, se trasladaron posteriormente a valores numéricos arbitrarios para obtener la desviación. A saber: $P=\{0.2, 0.3, 0.4\}$; $I=\{5, 10, 15\}$ días

diferencia de otras herramientas propietarias como LSF (*Load Sharing Facility*) o PBS (*Portable Batch System*), SLURM proporciona por medio de unas pocas configuraciones básicas un enorme abanico de combinaciones para generar infraestructuras heterogéneas y eficientes, además de contar con una amplia comunidad dedicada a su soporte e incluso con multitud de ejemplos que implementan escenarios de alta disponibilidad e integración de GPUs. En [30] se lleva a cabo una comparativa de varios representantes de este tipo de sistemas, pudiendo apreciar un claro énfasis en la gran versatilidad del susodicho gestor.

MariaDB/MySQL

MariaDB [41] es un sistema gestor de bases de datos que se origina a raíz de una derivación de código abierto de MySQL, por lo que presenta características bastante similares a las de su antecesor. Sin embargo, pretende ser una versión renovada y de mayor comodidad, que busca mejorar la experiencia de usuario mediante una respuesta más intuitiva, en especial durante la fase de pruebas. En este proyecto se utilizará para la creación de la parte destinada al *accounting* o registro histórico de información sobre las cargas de trabajo y principales interacciones de los usuarios.

NVML (NVIDIA Management Library)

Se trata de una librería de Nvidia que incluye una herramienta de línea de comandos, denominada `nvidia-smi`, para la monitorización del estado de las tarjetas gráficas. Es necesaria de cara a garantizar la correcta integración de los controladores de estos dispositivos en aquellos servidores que tengan pensado emplearlas.

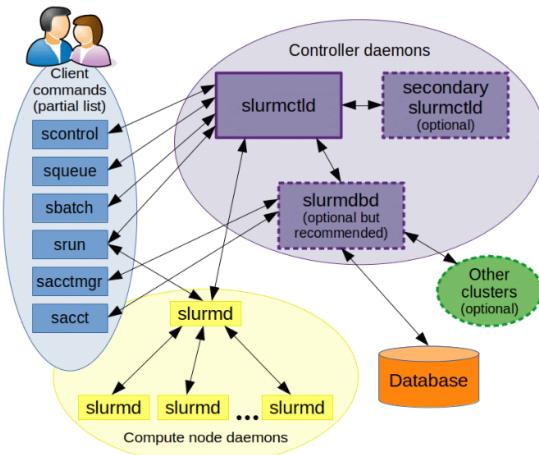


Figura 3.4: Estructura básica de SLURM

Prometheus

Engloba una serie de artefactos de código abierto para la recopilación y almacenamiento de métricas en formato de marcas temporales, útiles en procesos de monitorización de sistemas y en la creación de alertas asociadas. Presenta además un lenguaje propio (PromQL) para definir consultas de acceso a dicha información, la cual se extrae a partir de la salida que envían los exportadores. Estos son programas de diversa índole que utilizan *endpoints* para comunicar de forma periódica los datos necesarios al servidor central; pueden configurarse estáticamente o bien ser expuestos a través de un proceso de descubrimiento de servicios incluido en la propia herramienta. En concreto usaremos:

- **Node-exporter** [42], para obtener detalles genéricos sobre el estado de las máquinas.
- **Slurm-exporter** [43], con el que plasmar el contenido de las tablas de *accounting*, las colas o la salida de cualquiera de los comandos de diagnóstico.
- **Dcgm-exporter** [44], que nos permite recopilar métricas de las tarjetas gráficas de un nodo, pudiendo personalizar aquellas que se muestran mediante un fichero en formato CSV. Además cuenta con la opción de efectuar un despliegue orquestado mediante Docker o Kubernetes.

Es posible administrarla a través de una interfaz web, aunque lo habitual es utilizarla en conjunto con tecnologías como Grafana para darle una mejor apariencia y ofrecer una visión más detallada del estado de los recursos.

Grafana

Es una plataforma especializada en la visualización y análisis gráfico de fuentes de datos mediante la generación de paneles interactivos. Al igual que la anterior, contiene una interfaz web sobre la que se configura el aspecto de los *dashboards*. Es compatible con multitud de arquitecturas e incluso permite generar alertas personalizables, pudiendo estar acompañadas de métodos de envío de notificaciones como Slack o Microsoft Teams.

3.3.2 Desarrollo de los virtualizadores

Hyper-V

Hyper-V es el contendiente de Microsoft para la gestión de elementos virtualizados, preferiblemente con sistemas Windows aunque también soporta entornos Linux. Se trata de un hipervisor de tipo 1 centrado en el aislamiento por particiones (microkernelizado) [45], donde una de ellas actúa como un elemento padre soportando la infraestructura que interactúa con

los componentes físicos y sobre la que luego se crean los grupos con los sistemas operativos invitados. Por otra parte, para llevar a cabo dicha abstracción se utiliza un procesador compatible con la virtualización por *hardware* asistido, en la que los componentes nativos permiten la ejecución de instrucciones de las máquinas virtuales directamente sobre la CPU compartida [46].

Está orientado sobre todo al manejo de despliegues en entornos empresariales, y es posible encontrarlo como un producto *standalone* o como parte de los roles y características que se pueden instalar dentro de un equipo de Windows Server [45].

Proxmox

Es una solución de código abierto que combina la virtualización basada en *kernel* (KVM) y la tecnología propia de linux para la contenerización de procesos, denominada lxc [47]. A diferencia del candidato previo, Proxmox ofrece la posibilidad de controlar su entorno por interfaz web, además de una mayor libertad y escalabilidad en las tareas de gestión al no depender de paquetes propietarios, adaptándose fácilmente a los requisitos de ecosistemas PYME y emprendimientos con recursos limitados.

VMWare ESXi

Se trata de una implementación de *software* propietario con la particularidad de que no es una ramificación de otro sistema operativo, sino que mantiene una arquitectura confeccionada a medida para que todas las operaciones sean lo más óptimas y ligeras posible. Forma parte del paquete de vSphere, un conjunto de programas desarrollado por VMWare (y absorbido posteriormente por Broadcom) con el objetivo de proporcionar una experiencia altamente configurable y de gran profundidad para el despliegue de centros de datos. Al igual que en la plataforma previa, cuenta con una interfaz web para administrar cada servidor, aunque se puede llegar a centralizar gracias al *framework* de vCenter Server, que además permite integrar otros microservicios de la organización (e.g. vSAN).

Los servidores del CPD del CITIC se sustentan actualmente sobre este ecosistema, de modo que en el capítulo 5 también describiremos como sería la configuración en este entorno (de forma similar a los otros candidatos) y lo aprovecharemos como punto de partida en la comparación de rendimiento del capítulo 6.

3.3.3 Desarrollo de los benchmarks

Sysbench

Es una herramienta de evaluación de rendimiento enfocada en los elementos de CPU, memoria y disco, aunque también se puede utilizar con bases de datos relacionales.

Fio

Se centra en el análisis de las capacidades de lectura y escritura de unidades de almacenamiento, ya sean discos unitarios, sistemas RAID o sistemas de ficheros en red.

Iperf3

Se trata de un programa diseñado para medir las aptitudes de los enlaces de una red en términos de latencia y ancho de banda.

Tensorflow y Keras

Son módulos de python utilizados principalmente en el despliegue de modelos de aprendizaje automático y aprendizaje profundo. Los emplearemos para elaborar una carga de trabajo simple que verifique el nivel de procesamiento de las tarjetas gráficas.

CUDA/cuDNN

Se trata de librerías involucradas en la aceleración de cálculos de alto coste computacional y la agilización de los procesos de entrenamiento de redes neuronales. Son imprescindibles de cara a optimizar el rendimiento de los programas de este ámbito, por lo que casi siempre coexisten con paquetes como los mencionados anteriormente.

Gpu-burn

Consiste en una herramienta que realiza pruebas de estrés sobre las tarjetas gráficas en busca de fallos en el *hardware*, más concretamente de inestabilidades o fugas de memoria que no fuesen rectificadas por el módulo ECC (*Error Correcting Code*). Para ello lleva las **GPU** a los máximos valores de temperatura y capacidad computacional mediante secuencias en bucle de varios cálculos matriciales, ya que de esta forma se asegura de mantener un escenario propicio para la aparición de esta clase de errores.

En sí no es un **benchmark** orientado a mostrar la capacidad técnica real de estos componentes, pero es útil aplicarlo si se quiere validar su estado para una futura fase de producción.

3.3.4 Otras herramientas

MS Project

Es una herramienta de Microsoft destinada a la planificación de proyectos. Se ha utilizado principalmente para la creación del **diagrama de Gantt** y como referencia de cara al seguimiento del desarrollo.

Visual Studio Code

Es un editor de código fuente compatible con una gran cantidad de lenguajes de programación, cuyas funcionalidades base se pueden complementar mediante la agregación de extensiones. Se ha utilizado principalmente para programar y ejecutar en python el *script* destinado a la extracción de datos en la fase de análisis comparativo.

Anaconda

Representa una distribución de python para la gestión de paquetes y sus dependencias de forma sencilla a través de entornos virtuales aislados. En el sector [HPC](#) resulta bastante conveniente, puesto que permite instalar simultáneamente varias versiones específicas de programas según las necesidades de los usuarios, sin que por ello existan problemas de compatibilidad entre sus versiones.

Capítulo 4

Despliegue del clúster

TRAS haber profundizado en los principales aspectos que influencian nuestro desarrollo y cómo se llevará a cabo su planificación, pasamos a exponer el proceso seguido durante su ciclo de vida, empezando por la parte del sistema de colas. En primer lugar daremos una breve descripción sobre el uso de los recursos computacionales en el [CPD](#) de la entidad, para posteriormente mostrar los elementos necesarios en la creación del clúster y la forma en la que se ha acometido cada tarea.

4.1 Análisis de la infraestructura actual

El centro de datos del [CITIC](#) mantiene un conglomerado de virtualización que incluye 13 servidores y 38 [GPUs](#), de las cuales 13 son de uso exclusivo de alguno de los grupos de investigación y 16 disponen de una o varias máquinas asignadas. Por otra parte, la organización estableció recientemente una política de restricción de uso sobre las tarjetas gráficas, provocando que sea necesario renovar el período de ocupación mensualmente y obligando a mantener un registro de estos intervalos mediante un formato similar al señalado en la figura 4.1.

A fecha de inicio del proyecto se pudieron observar un total de 34 peticiones activas, donde cerca del 80% han extendido su duración al menos una vez. A mayores, alrededor de un 20% de las máquinas están vinculadas a tarjetas propias de cada colectivo investigador (i.e etiquetadas como asignaciones permanentes) y del resto un 15% se destinaron a servicios en producción. También se estimó que un 77% del histórico de solicitudes establece una ocupación completa del recurso reservado, lo cual denota que son propensos a ejecutar cargas de trabajo exigentes.

Por lo tanto, se puede concluir que existe una demanda notable de estos componentes y una clara tendencia a prolongar el tiempo de dichas reservas. Dada la criticidad de estos proyectos en el ecosistema [TI](#) de la organización, la agregación de una arquitectura independiente para el desempeño de estas actividades se convierte entonces en una buena oportunidad de

negocio.

Permanentes	Historico	En creacion	Servicios producción	Revisar [Apagado]	Nuevas renovaciones
Bombero Maquina Virtual					
20240606	Bayonder23-L40-1	25/11/2024	01/06/2054	1	7700
20210617	Bayonder18-RTX8000-2	25/11/2024	01/06/2054	1	7700
20220324	Bayonder16-RTX A6000-2	25/11/2024	01/06/2054	1	7700
20220324	Bayonder18-RTX A6000-1	25/11/2024	01/06/2054	1	7700
20220325	Bayonder21-RTX A6000-1	24/11/2024	02/06/2054	1	7700
20220325	Bayonder21-RTX A6000-2	27/11/2024	03/06/2054	1	7700
20220329	Bayonder17-A100-1	27/11/2024	03/06/2054	1	7700
20221229	Bayonder14-A100-2	27/11/2024	03/06/2054	1	7700
20220712	Bayonder14-A100-1	27/11/2024	03/06/2054	1	7700
20241024	Bayonder22-L40-1	25/11/2024	25/12/2045	1	7700
20241025	Bayonder22-L40-2	25/11/2024	25/12/2045	1	7700
20241026	Bayonder22-L40-3	25/11/2024	25/12/2045	1	7700
20230911	Bayonder17-A100-4	28/11/2024	07/01/2053	1/4	30
20200424	Bayonder10-Tesla P40-1	25/11/2024	06/01/2053	1	30
20200424	Bayonder11-Tesla P40-1	25/11/2024	06/01/2053	1	30
20210509	Bayonder11-Tesla P40-2	25/11/2024	01/06/2054	1	7700
20230110	Bayonder12-Tesla P40-1	05/02/2025	19/03/2025	1	30

Figura 4.1: Fragmento del histórico de solicitudes de GPU

4.2 Requisitos previos

Para poder montar un clúster en [SLURM](#) es preciso que todos los nodos involucrados contengan un sistema operativo de tipo *unix* [40], siendo Ubuntu la opción por la que nos hemos decantado. Concretamente utilizaremos la versión *Server 22.04 LTS*, ya que se encuentra bastante asentada en el panorama informático y es relativamente reciente. En consecuencia, cualquier programa partícipe del despliegue ha de ser compatible con ella.

Por otro lado, debemos contar con una zona común de almacenamiento para que los usuarios finales puedan transferir de forma eficiente tanto los datos asociados a sus ejecuciones como las configuraciones del clúster a los nodos. Esto también nos lleva a plantear la segregación de las conexiones en varias redes: una de gestión, una de almacenamiento y, opcionalmente, una para el cómputo de tareas en paralelo.

Otro aspecto clave es la recopilación de los datos resultantes de las ejecuciones, para lo cual se integrará una base de datos en MySQL (al ser la única opción compatible con esta tecnología) que interactuará con el componente central del clúster.

4.3 Diseño inicial y topología

Teniendo en mente las condiciones predispuestas, la estructura a crear estaría formada por al menos un nodo controlador, un nodo de cómputo¹, un nodo para el almacenamiento compartido, un nodo para la base de datos y un nodo de login con el que trasmisitir cómodamente las peticiones al clúster. En la figura 4.2 se aprecia una posible disposición de las comunicaciones y servicios.

¹ En el anteproyecto se comentó que idealmente habría dos servidores de este tipo, pero por cuestiones logísticas y otros factores externos a este proyecto nos hemos visto forzados a reducir el despliegue.

Con ello logramos una separación adecuada de las responsabilidades, aislando los datos de la arquitectura y permitiendo su recuperación independientemente de lo que suceda en el entorno. En lo que respecta a las conexiones, todos los nodos menos el de almacenamiento tienen designada una interfaz en la red de gestión, y tanto este último como el de *accounting* cuentan con una interfaz en la red de almacenamiento.

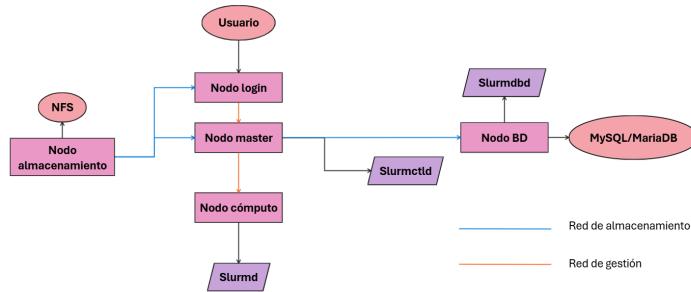


Figura 4.2: Esquema topológico y conexiones entre nodos

4.4 Implementación

4.4.1 Usuarios y sincronización

Antes de poder siquiera desenvolver la parte del planificador, debemos asegurar que existe conectividad entre los nodos del clúster y que sus comunicaciones en la red de gestión permanezcan sincronizadas e íntegras. Para ello, se precisa incluir en todas las máquinas los usuarios `slurm` y `munge`², donde el primero será el identificador principal de las operaciones y el segundo se utilizará para administrar claves de autenticación. También es recomendable editar el fichero `/etc/hosts` para darle un nombre acorde a cada máquina, puesto que facilitará en la configuración de `SLURM` el reconocimiento de los elementos responsables de cada tarea.

```

1 sudo adduser -u 1111 munge --disabled-password --gecos ""
2 sudo adduser -u 1121 slurm --disabled-password --gecos ""

```

En cuanto a los intervalos de tiempo, podemos configurar el nodo maestro como un servidor NTP. Para ello, habiendo instalado el paquete correspondiente, incluimos en el archivo `ntp.conf` las siguientes líneas:

```

1 # Permitir clientes en la red interna
2 restrict <red_gestión> mask 255.255.255.0 nomodify notrap
3 restrict 127.0.0.1
4 restrict ::1

```

² Los UIDs empleados en la inserción pueden ser distintos, siempre y cuando no estén reservados por el sistema.

Dado que puede haber acceso a Internet, es útil mantener al menos una referencia de los servidores públicos para garantizar que la sincronización sea correcta. Por otro lado, en los nodos que van a actuar de cliente se añade en su propio `ntp.conf` la línea `server <IP_nodo_maestro> iburst`. Una vez reiniciado el servicio en ambos extremos, invitamos a `date`³ y `ntpq -p` (Figura 4.3) desde uno de los nodos cliente para revisar la hora y el estado de la conexión.

```
bdacc@bdaccslurmtest:~$ date
lun 14 abr 2025 14:06:23 CEST
bdacc@bdaccslurmtest:~$ ntpq -p
      remote          refid      st t when poll reach   delay    offset  jitter
===== 
+maestro.slurmte 185.125.190.58    3 u    33 1024  377    0.248  +1.063  0.967
  ntp.ubuntu.com .POOL.        16 p    - 64    0    0.000  +0.000  0.000
-185.125.190.56  201.68.88.106   2 u    54 1024  377   47.681  -0.154  1.360
+185.125.190.58  79.243.60.50    2 u    867 1024  377   44.308  +1.029  1.759
alphyn.canonica 132.163.96.1     2 u    88 1024  377   107.600  +1.891  4.732
+185.125.190.57  17.253.28.251   2 u    191 1024  377   47.587  -0.151  1.411
bdacc@bdaccslurmtest:~$
```

Figura 4.3: Sincronización de cliente NTP

4.4.2 Autenticación de comunicaciones

Otro de los requisitos que se indican en la documentación es disponer de algún método para verificar la identidad de los procesos transmitidos entre los nodos del clúster. El mecanismo por defecto es `munge`, un servicio de autenticación centrado en la generación de credenciales codificadas [48], similar a un procedimiento de firma y cifrado de un documento, con las que se reconoce al usuario emisor de la petición de forma transparente.

Para su correcto funcionamiento se precisa que todos los nodos posean la misma clave de autenticación, la cual ha de ubicarse en `/etc/munge/munge.key` y contar con permisos de sólo lectura. Por cuestiones de practicidad utilizaremos como base la clave del nodo controlador⁴. En la figura 4.4 se verifica que el uso de las claves están funcionando correctamente, probando a mayores la codificación en ambas direcciones de un *token* temporal.

```
1 # En el nodo controlador
2 sudo apt-get install libmunge-dev libmunge2 munge -y
3 sudo systemctl enable munge
4 sudo systemctl start munge
5 munge -n | unmunge | grep STATUS # Verifica si se pueden codificar
       y descodificar credenciales. Debería salir Success
6 sudo cp /etc/munge/munge.key /slurmsharing/
7 sudo chown munge /slurmsharing/munge.key
8 sudo chmod 400 /slurmsharing/munge.key
```

³ Si la zona horaria no coincide, se puede modificar con el comando `sudo timedatectl set-timezone <Zona/Region>`

⁴ Salvo que se indique lo contrario, cualquier comando sucesivo será invocado desde este servidor.

```

9
10 # En el resto de nodos
11 sudo apt-get install libmunge-dev libmunge2 munge
12 sudo cp /slurmsharing/munge.key /etc/munge/munge.key
13 sudo systemctl enable munge
14 sudo systemctl start munge
15 munge -n | unmunge | grep STATUS

```

```

maestro@maestroslurmtest:~$ munge -n | unmunge | grep STATUS
STATUS: Success (0)
maestro@maestroslurmtest:~$ munge -n -t 15 | ssh login@login unmunge
login@login's password:
STATUS: Success (0)
ENCODE_HOST: maestro.slurmtest.local (10.56.████)
ENCODE_TIME: 2025-05-23 09:42:01 +0200 (1747986121)
DECODE_TIME: 2025-05-23 09:42:05 +0200 (1747986125)
ITL: 15
CIPHER: aes128 (4)
MAC: sha256 (5)
ZIP: none (0)
UID: login (1000)
GID: login (1000)
LENGTH: 0

maestro@maestroslurmtest:~$ ssh login@login munge -n -t 15 | unmunge
login@login's password:
STATUS: Success (0)
ENCODE_HOST: login.slurmtest.local (10.56.████)
ENCODE_TIME: 2025-05-23 09:42:13 +0200 (1747986133)
DECODE_TIME: 2025-05-23 09:42:13 +0200 (1747986133)
ITL: 15
CIPHER: aes128 (4)
MAC: sha256 (5)
ZIP: none (0)
UID: maestro (1000)
GID: maestro (1000)
LENGTH: 0

maestro@maestroslurmtest:~$ █

```

Figura 4.4: Resultado tras probar munge

4.4.3 Almacenamiento compartido

Para desplegar esta parte usaremos el protocolo **NFS**, cuyo objetivo es exponer un directorio por red a cualquier máquina a fin de convertirse en un sistema de ficheros remoto. Su comportamiento también se basa en una relación cliente-servidor, que se configura llevando a cabo estos pasos:

Nodo servidor

```

1 sudo apt-get update
2 sudo apt-get install nfs-kernel-server
3 sudo mkdir /<directorio> -p
4 sudo chown <usuario_alm>:<usuario_alm> <directorio>
5 # usuario_alm := usuario principal de este nodo
6
7 # Incluir para cada receptor en /etc/exports:

```

```

8 <directorio> <IP_destino> (rw,sync,no_root_squash,no_subtree_check)
9
10 sudo systemctl restart nfs-kernel-server

```

Nodos cliente

```

1 sudo apt-get update
2 sudo apt-get install nfs-common
3 sudo mkdir -p <directorio>
4 sudo mount <IP_nodo_alm>:<directorio> /<directorio>
5
6 # Para que persista tras reinicio, añadir a /etc/fstab:
7 <IP_nodo_alm>:<directorio> /<directorio> nfs auto,timeo=14,intr 0 0
8
9 sudo chown <usuario_worker>:<usuario_worker> /<directorio>/
10 # usuario_worker := usuario principal de cada cliente.

```

De esta forma, dispondremos de un directorio (/slurmsharing) para albergar los archivos necesarios en la instalación del entorno y otro dedicado a mantener los códigos fuente de los programas a utilizar en las pruebas, los scripts que se encargan de enviar las peticiones al clúster y los ficheros con el resultado de estas (/datasharing). En la figura 4.5 se observa el resultado de realizar estas operaciones en uno de los nodos.

Filesystem	Size	Used	Avail	Use%	Mounted on
tmpfs	392M	1,3M	390M	1%	/run
/dev/mapper/ubuntu--vg-ubuntu--lv	39G	12G	25G	31%	/
tmpfs	2,0G	0	2,0G	0%	/dev/shm
tmpfs	5,0M	0	5,0M	0%	/run/lock
tmpfs	4,0M	0	4,0M	0%	/sys/fs/cgroup
/dev/sda2	2,0G	245M	1,6G	14%	/boot
10.56.██:/slurmsharing	98G	30G	64G	32%	/slurmsharing
10.56.██:/datasharing	98G	30G	64G	32%	/datasharing
tmpfs	392M	4,0K	392M	1%	/run/user/1000

Figura 4.5: Resultado del montaje de NFS

4.4.4 Compilación del paquete

Al llevar a cabo una construcción manual del paquete, es necesario obtener desde la página oficial el archivo comprimido con la versión de SLURM deseada (en nuestro caso la 23.11.4) y especificar la estructuración de las dependencias en el código fuente:

```

1 cd /slurmsharing
2 wget https://download.schedmd.com/slurm/slurm-23.11.4.tar.bz2
3 bunzip2 -k slurm-23.11.4.tar.bz2
4 tar -xvf slurm-23.11.4.tar

```

```
5  
6 # En el nodo de accounting  
7 sudo apt-get install python3 gcc make openssl ruby ruby-dev  
    libpam0g-dev libmariadb-dev mariadb-server build-essential  
    libssl-dev numactl hwloc libmunge-dev man2html lua5.3 -y  
8  
9 cd slurm-23.11.4/  
10 ./configure --prefix=/tmp/slurm-build --sysconfdir=/etc/slurm  
    --enable-pam --with-pam_dir=/lib/x86_64-linux-gnu/security/  
    --without-shared-libslurm  
11 make  
12 make contrib  
13 make install  
14 cd ..
```

Donde *prefix* indica el lugar para almacenar el compilado resultante y *sysconfdir* señala el directorio donde se albergarán los archivos que dictaminan el funcionamiento del entorno. Las demás opciones sirven para habilitar la compatibilidad con el bloque PAM del sistema operativo, establecer la ruta en la que se localizan los módulos de seguridad correspondientes y evitar que el paquete se utilice como una librería compartida.

Otro aspecto a reseñar son los requisitos de la construcción del código. SLURM utiliza un sistema de módulos en forma de *plug-in* para detectar diferentes programas (entre ellos la base de datos) y hacerse una idea de cómo estos han de funcionar. Por este motivo se ha escogido el servidor de *accounting* para realizar dicho proceso, ya que nos evita tener que llevar a cabo instalaciones redundantes o innecesarias en los otros nodos.

4.4.5 Instalación

Una vez tenemos el código compilado, lo siguiente es generar el paquete *debian* correspondiente para que pueda ser incorporado por el resto de servidores.

```
1 sudo apt install -y ruby ruby-dev  
2 sudo gem install --no-document fpm  
3  
4 sudo fpm -s dir -t deb -v 1.0 -n slurm-23.11.4 --prefix=/usr -C  
    /tmp/slurm-build/ .  
5  
6 # Para instalarlo en cada nodo  
7 sudo dpkg -i slurm-23.11.4_1.0_amd64.deb
```

Asimismo, creamos algunos directorios que serán útiles en el registro de *logs* y la gestión de los servicios en segundo plano:

```
1 # Para nodo controlador
```

```

2 sudo mkdir -p /etc/slurm /var/log/slurm /var/spool/slurm/ctld
   /var/spool/slurm/d
3 sudo chown slurm /var/log/slurm /var/spool/slurm/ctld /var/spool/slurm/d
4
5 # Para nodo de accounting
6 sudo mkdir -p /var/log/slurm
7 sudo chown slurm /var/log/slurm
8
9 # Para nodo de cómputo
10 sudo mkdir -p /etc/slurm /var/log/slurm /var/spool/slurm/d
11 sudo chown slurm /var/log/slurm /var/spool/slurm/d

```

El directorio `/etc/slurm` ha de estar presente en todos los miembros del clúster y contendrá únicamente los ficheros de configuración que se adecúen a los diferentes roles. Así pues, la herramienta los desglosa principalmente en tres archivos: `slurm.conf`, que define la organización del clúster, los atributos de cada elemento y otros rasgos sobre las interacciones entre servicios; `slurmdbd.conf`, que determina las características concretas de la base de datos y sus credenciales de conexión; y `gres.conf`, en el que se especifica la asociación entre nodos y tarjetas gráficas.

A mayores se precisa de ficheros `*.service` para permitir la ejecución de los aplicativos en segundo plano (ver figura 4.2), los cuales se copiarán al directorio `/etc/systemd/system` de los nodos siguiendo una correspondencia análoga al párrafo anterior. La descripción completa de ambos tipos de archivos se puede consultar en el Anexo A.1, dado que en este capítulo nos dedicaremos a comentar los aspectos más destacados.

Configuraciones básicas

La esquematización del clúster se puede condensar en las siguientes variables del fichero `slurm.conf`:

- **ClusterName:** Indica el nombre que le damos al clúster, en nuestro caso `citic-slurmtest`.
- **ControlMachine:** El nombre completo o `hostname` de la máquina que actuará de controlador.
- **SlurmUser, SlurmctldPort y SlurmdPort:** Usuario bajo el que se ejecutan los servicios en segundo plano, junto a los puertos que exponen el controlador y los nodos de cómputo para realizar las comunicaciones. Se han dejado los valores por defecto (slurm, 6817 y 6818 respectivamente).
- **StateSaveLocation:** Ruta donde se guardan instantáneas del estado del nodo controlador, para que en caso de haber uno de reserva pueda asumir fácilmente sus funciones a partir de la información cacheada.

- **SchedulerType**: Marca el algoritmo a utilizar en la ordenación y procesamiento de las colas. Le indicamos *sched/backfill* para tratar de priorizar los trabajos más livianos.
- **SelectType, SelectTypeParameters**: Define si las asignaciones de recursos para cada petición se hacen en modo exclusivo (*select/linear*) o si se permite planificar varios trabajos a la vez en un mismo nodo (*select/cons_tres*). Se ha seleccionado la segunda opción, ya que buscamos paralelizar el mayor número de trabajos posible de cara a minimizar los tiempos de espera. Por otro lado, los *parameters* establecen los recursos sobre los que se debe monitorizar dicha partición, pudiendo escoger a nivel de núcleo de CPU, memoria, memoria/núcleo e incluso por GPU.
- **JobAcctGatherType**: Señala la procedencia de la información recopilada sobre el consumo de componentes por cada trabajo. Se pueden utilizar como referencia la tabla de procesos del propio sistema o bien recurrir a los *cgroups*, una característica del kernel de Linux para la restricción y aislamiento de recursos en un conjunto de procesos. En nuestro caso nos hemos decantado por esta última opción, cuya activación indicaremos más adelante.
- **AccountingStorage^{*}**: En este punto se incluyen todas las opciones relacionadas con el registro de trabajos, cuentas, asociaciones de recursos y sus restricciones en la base de datos.
- **NodeName, PartitionName**: Sirven para determinar los miembros del clúster y la forma en la que se agrupan, además de indicar sus especificaciones. Estas últimas se pueden obtener a través de los comandos `lscpu` y `free -m`, aunque también se puede utilizar `sudo slurmd -C` para obtener directamente el formato de la entrada a introducir (Figura 4.6).

```
(base) compl@complslurmtest:~$ sudo slurmd -C
[sudo] password for compl:
NodeName=complslurmtest CPUs=24 Boards=1 SocketsPerBoard=2 CoresPerSocket=6 ThreadsPerCore=2 RealMemory=515877
UpTime=9-20:09:51
(base) compl@complslurmtest:~$ 
```

Figura 4.6: Ejemplo de salida del comando `slurmd -C`

Creación de la base de datos

En el nodo que vayamos a utilizar para el *accounting* deben estar instalados los paquetes de MySQL/MariaDB, para posteriormente establecer las configuraciones de la base de datos junto a sus credenciales de acceso, donde se incluirá como usuario principal al mismo encargado de administrar el clúster (i.e. `slurm`).

```

1 sudo systemctl enable mysql
2 sudo systemctl start mysql
3
4 sudo mysql -u root
5 create database slurm_acct_db; # Es el nombre que SLURM utiliza por
     defecto para referirse a ella
6 create user 'slurm'@'localhost';
7 set password for 'slurm'@'localhost' = password('<contraseña>');
8 grant usage on *.* to 'slurm'@'localhost';
9 grant all privileges on slurm_acct_db.* to 'slurm'@'localhost';
10 flush privileges; # Aplica los permisos otorgados
11 exit

```

Asimismo, en los archivos de `slurm.conf` y `slurmdbd.conf` se deben reflejar adecuadamente los cambios realizados anteriormente. A continuación se muestra un fragmento de ambos con las variables afectadas:

```

1 # En slurm.conf
2 AccountingStorageType=accounting_storage/slurmdbd
3 AccountingStorageHost=localhost
4 AccountingStoragePass=/var/run/munge/munge.socket.2
5 AccountingStorageUser=slurm
6
7 # En slurmdbd.conf
8 DbdAddr=<IP_almacenamiento_accounting>
9 DbdHost=localhost
10 SlurmUser=slurm
11 StorageType=accounting_storage/mysql
12 StorageHost=bdacc
13 StoragePass=<insertar_contraseña>
14 StorageUser=slurm
15 StorageLoc=slurm_acct_db

```

Compatibilidad con GPUs

Para habilitar esta parte debemos primero instalar una versión adecuada de los drivers de nvidia; de no saber cuáles son aptas para nuestro sistema operativo, se puede utilizar el comando `ubuntu-drivers devices` (Figura 4.7) y escoger la que mejor se adapte a la situación. Posteriormente se reinicia el servidor para aplicar los cambios, y al invocar `nvidia-smi` podremos ver una tabla con las GPU vinculadas al host (Figura 4.8), acompañadas de otros valores como el consumo energético, la cantidad de vRAM que está siendo ocupada en ese instante y los procesos que están ejecutando.

Ahora bien, si deseamos que puedan ser utilizadas en los trabajos del clúster tenemos que incluir las siguientes líneas en los ficheros de `slurm.conf` y `gres.conf`:

```

1 # En slurm.conf
2 GresTypes=gpu # Incluir también Gres=gpu:<número> en las
   especificaciones de los nodos pertinentes.
3
4 # En gres.conf, por cada tarjeta disponible:
5 Nodename=<nombre> Name=gpu File=/dev/nvidia<número>

```

```
(slurm-env) compl@complslurmtest:/datasharing/jobs/benchmarks/tensorflow$ ubuntu-drivers devices
ERROR:root:play command not found
== /sys/devices/pci0000:00/0000:00:03.0/0000:02:00.0 ==
modalias : pci:v000010DEd00002230s0v000010DEd00001459bc03sc02i00
vendor : NVIDIA Corporation
model : GA102GL [RTX A6000]
driver : nvidia-driver-570-server-open - distro non-free
driver : nvidia-driver-470-server - distro non-free
driver : nvidia-driver-570-server - distro non-free
driver : nvidia-driver-550-open - distro non-free
driver : nvidia-driver-535-server-open - distro non-free
driver : nvidia-driver-550 - distro non-free recommended
driver : nvidia-driver-535 - distro non-free
driver : nvidia-driver-545-open - distro non-free
driver : nvidia-driver-535-open - distro non-free
driver : nvidia-driver-470-server - distro non-free
driver : nvidia-driver-535-server-open - distro non-free
driver : nvidia-driver-535 - distro non-free
driver : xserver-xorg-video-nouveau - distro free builtin
(slurm-env) compl@complslurmtest:/datasharing/jobs/benchmarks/tensorflow$ 
```

Figura 4.7: Lista de controladores de NVIDIA disponibles

```
compl@complslurmtest:~$ nvidia-smi
Mon Mar 24 13:15:35 2025
+-----+
| NVIDIA-SMI 535.183.01      Driver Version: 535.183.01    CUDA Version: 12.2 |
+-----+
| GPU  Name        Persistence-M | Bus-Id     Disp.A | Volatile Uncorr. ECC | | |
| Fan  Temp     Perf  Pwr:Usage/Cap | Memory-Usage | GPU-Util Compute M. |
|          |          |              |              | MIG M. |
+-----+
| 0  NVIDIA RTX A6000     Off  00000000:02:00.0 Off |           0 |      Default | N/A |
| 30% 38C   P8          7W / 300W 0MiB / 46068MiB |      0% |      Default | N/A |
+-----+
| 1  NVIDIA RTX A6000     Off  00000000:01:00.0 Off |           0 |      Default | N/A |
| 30% 37C   P8          8W / 300W 0MiB / 46068MiB |      0% |      Default | N/A |
+-----+
| Processes:
|  GPU  GI CI PID  Type  Process name          GPU Memory |
|    ID  ID   ID   ID    ID      Usage          Usage
+-----+
| No running processes found
+-----+
compl@complslurmtest:~$ 
```

Figura 4.8: Salida del comando nvidia-smi

4.5 Lanzamiento y pruebas

Habiendo completado todos los pasos anteriores, reiniciamos las máquinas y arrancamos los servicios asociados a cada una, además de registrar al clúster como entidad en la sección de *accounting* mediante `sudo sacctmgr add cluster <ClusterName>`. A partir de este punto la infraestructura está preparada para recibir nuevas peticiones; sintetizaremos las sentencias más utilizadas, la forma en la que se genera una solicitud y cómo podemos supervisar

su progresión.

4.5.1 Comandos principales

Si bien la herramienta contiene multitud de comodidades, es probable que para un usuario final muchas de estas opciones le resulten desconcertantes a primera vista o no tenga claro cuál es la forma de proceder. Por este motivo nos ceñiremos a explicar las mostradas en la lista de la Figura 3.4, pues consideramos que son las idóneas para un manejo rutinario de estos sistemas, tanto por parte del cliente como de un administrador.

De cliente

srun: Es el comando principal para alojar una ejecución en alguno de los nodos que se encuentren disponibles, pudiendo introducir argumentos como los metadatos de la actividad, las especificaciones de recursos requeridas e incluso en qué máquinas particulares se desea llevar a cabo la ejecución. De forma análoga, contamos con el comando `scancel` para la interrupción de trabajos a nivel de cuenta, nodo o incluso de partición.

sbatch: Permite programar una solicitud de carga de trabajo, que el planificador procesará cuando se disponga de los recursos determinados. Puede contener una o varias sentencias de `srun` con tareas a realizar.

squeue: Ofrece una visión en forma de tabla de las solicitudes enviadas, su estado y, en caso de no estar siendo procesadas, la razón de su emplazamiento.

De administrador

scontrol: Se utiliza para modificar la configuración del entorno sin necesidad de acceder al fichero de `slurm.conf`, o bien para aplicar cualquier cambio que haya quedado pendiente del mismo. También permite visualizar el estado de diversos elementos como particiones o trabajos en curso invocando la orden `show <entidad>`.

sacct: Sirve para recopilar información acerca de las actividades completadas. Por defecto sólo extrae aquellas que se han ejecutado en el mismo día, pudiendo también filtrarlas en base a un intervalo de tiempo, por estado o por ciertos grupos de usuarios y particiones, entre otros factores.

sacctmgr: Homólogo del comando anterior para la gestión de las entidades involucradas en la base de datos y configurar reglas de calidad de servicio que permitan restringir la asignación total de cada cuenta del clúster o bien la de los propios usuarios de la máquina.

4.5.2 Creación de una carga de trabajo

Al elaborar un *script* de petición al clúster, podemos incorporar diversas opciones que ajusten las condiciones necesarias para su compleción (acompañadas siempre del fragmento `#SBATCH`). El bloque siguiente muestra un ejemplo simplificado de su estructura:

```

1 #!/bin/bash
2 #SBATCH --job-name=<nombre>
3 #SBATCH --output=<archivo>.out    # Fichero con la salida del
4      programa.
5 #SBATCH --error=<archivo>.err
6 #SBATCH --time=<HH:mm:ss>          # Tiempo máximo de ejecución.
7 #SBATCH --nodes=<número>
8 #SBATCH --ntasks=<número>         # Steps o iteraciones del trabajo.
9      Superior a 1 => paralelizado.
10 #SBATCH --cpus-per-task=<número>
11 #SBATCH --gres=gpu:<id_gpu>
12 #SBATCH --mem=<número_y_magnitud>  # e.g. 8G => 8 GB
13 #SBATCH --partition=<partición>
14
15 module load <lista_módulos>
<Comando/s_a_ejecutar>...

```

La sentencia para cargar módulos se debe incluir siempre y cuando se esté utilizando alguna herramienta como lmod para manejar las versiones del *software* ejecutado por un usuario. Puesto que estamos siguiendo un enfoque de pruebas decidimos que lo más sencillo era mantener una instalación global de los paquetes, por lo que no precisaríamos agregar dicha línea. No obstante, para aquellas librerías importadas en códigos python o similares se ha recurrido a un entorno virtual en anaconda, pues nos facilita en gran medida la gestión de dependencias.

Por otro lado, en el anexo A.3 se encuentran los programas que se han confeccionado para medir el rendimiento del clúster (cuyos resultados se analizarán en el capítulo 6), a saber:

- Una ejecución para evaluar la capacidad de procesamiento a nivel de CPU, memoria, disco y red⁵, que por comodidad pasaremos a nombrar como ***compbench***. Será además el principal referente a la hora de establecer el contraste con el resto de plataformas.
- Un código escrito en python para el entrenamiento de un modelo sencillo con keras y tensorflow, que llamaremos ***tf-train***.
- El programa ***gpu-burn***, con el que llevar a cabo varias pruebas de estrés sobre las GPU.

⁵ Esta última simula una interacción cliente-servidor para determinar el máximo ancho de banda entre el nodo maestro y un nodo de cómputo.

En cualquier caso, dichos programas también nos han servido para recrear ciertos conflictos, observar cómo se gestionan dentro de la infraestructura y de paso poner en práctica las características listadas a lo largo del capítulo.

4.5.3 Escenario 1: Priorización por backfilling

En el capítulo 2 hablábamos de la relevancia de aplicar estrategias de optimización en el encolado de trabajos, siendo el *backfilling* una de las más empleadas en este ámbito al impedir que trabajos con exigencias laxas experimenten problemas de inanición.

Para ver los efectos de este comportamiento programamos varias cargas de diferentes requisitos. En particular hemos utilizado *compbench* y *tf-train*, ya que en ambas hemos considerado una cantidad de núcleos de CPU y memoria distintas. En la Figura 4.9 se muestra como el orden en el que se despachan los trabajos no es secuencial, sino que busca que se ejecute antes el programa de *tf-train* al requerir un menor número de núcleos (4 frente a 16).

```
login@loginslurmtest:/datasharing/jobs/benchmarks$ squeue
      JOBID PARTITION     NAME   USER ST      TIME  NODES NODELIST(REASON)
        68  compute benchmar login PD    0:00      1 (Resources)
        70  compute tfflowjob login PD    0:00      1 (Priority)
        69  compute tfflowjob login R     0:12      1 complslurmtest
        67  compute tfflowjob login R     0:34      1 complslurmtest
        66  compute benchmar login R     0:42      1 complslurmtest
login@loginslurmtest:/datasharing/jobs/benchmarks$ squeue
      JOBID PARTITION     NAME   USER ST      TIME  NODES NODELIST(REASON)
        68  compute benchmar login PD    0:00      1 (Resources)
        70  compute tfflowjob login R     0:03      1 complslurmtest
        69  compute tfflowjob login R     0:33      1 complslurmtest
        66  compute benchmar login R    1:03      1 complslurmtest
login@loginslurmtest:/datasharing/jobs/benchmarks$
```

Figura 4.9: Demostración de la política de backfill

4.5.4 Escenario 2: Limitación de recursos global

Con el fin de asegurar un correcto aislamiento de los recursos para cada proceso, SLURM admite la integración de *cgroups*, un mecanismo que ofrecen los sistemas Linux para supervisar y limitar el uso de estos elementos. Concretamente se basa en una organización por directorios en forma de árbol, donde se incluye a su vez una jerarquía por cada subsistema o grupo de componentes considerado. De esta forma, cada vez que un nodo de cómputo esté procesando una petición se generará dinámicamente una entrada específica por usuario, trabajo y tareas involucradas en ese instante.

Según la documentación, este comportamiento se versiona a través de tres *plug-in*: **proc-track/cgroup** (seguimiento y gestión del estado de los procesos), **task/cgroup** (aplicación de restricciones sobre recursos en cada trabajo o tarea) y **jobacct_gather/cgroup** (recopilación de estadísticas de las ejecuciones).

Por otro lado, habilitar esta característica requiere de dos archivos: `cgroup.conf`, donde se indican propiamente los componentes a gestionar, y `allowed_devices_file.conf`, en el que se establecen las particiones o dispositivos sobre los que puede actuar. Ambos deben ser copiados al directorio `/etc/slurm` de los nodos de cómputo, además de añadir en la configuración de su `grub` o gestor de arranque la siguiente cadena:

```
1 GRUB_CMDLINE_LINUX_DEFAULT="cgroup_enable=memory
    systemd.unified_cgroup_hierarchy=0"
```

La razón detrás de estos valores reside en utilizar la versión 1 de la herramienta (`cgroups/v1`), pues con ella nos aseguramos su compatibilidad dentro del clúster. Asimismo, insertamos las variables pertinentes al fichero de `slurm.conf`.

```
1 ProctrackType=proctrack/cgroup
2 TaskPlugin=task/cgroup
3
4 # Opcionalmente, para registro de métricas:
5 JobAcctGatherType=job_acct_gather/cgroup
```

Para verificar su funcionamiento mandamos ejecutar la tarea de la figura 4.10, en la que se lanza una sesión por terminal dentro del nodo de cómputo. De esta forma vemos cómo se ha llegado a crear una entrada en cgroups para el usuario responsable de la petición, dentro del apartado de CPU. Además, al revisar la afinidad de los procesadores se ve que está limitado a 4 núcleos, cuyos identificadores se corresponden con los asignados al primer procesador.

```
worker@loginslurmtest:/home/login$ srun --account=citic --partition=compute --cpus-per-task=4 --pty bash
slurmstepd: error: couldn't chdir to '/home/login': No such file or directory: going to /tmp instead
slurmstepd: error: couldn't chdir to '/home/login': No such file or directory: going to /tmp instead
(base) worker@complslurmtest:/tmp$ 
(base) worker@complslurmtest:/tmp$ lscpu | grep "CPU(s)"
CPU(s):                          24
On-line CPU(s) list:            0-23
NUMA node0 CPU(s):              0-5,12-17
NUMA node1 CPU(s):              6-11,18-23
(base) worker@complslurmtest:/tmp$ 
(base) worker@complslurmtest:/tmp$ nproc
4
(base) worker@complslurmtest:/tmp$ taskset -pc $$
pid 2747's current affinity list: 0,1,12,13
(base) worker@complslurmtest:/tmp$ 
(base) worker@complslurmtest:/tmp$ cat /sys/fs/cgroup/cpu
cpu/
cpuacct/
cpu,cpuacct/ cpuset/
(base) worker@complslurmtest:/tmp$ cat /sys/fs/cgroup/cpuset/slurm/uid_5000/job_132/cpus
0-1,12-13
(base) worker@complslurmtest:/tmp$ 
```

Figura 4.10: Prueba de cgroups sobre núcleos de CPU

4.5.5 Escenario 3: Limitación de recursos por cuenta

Además de las restricciones anteriores, es posible establecer cuotas de recursos tanto por cuenta de la base de datos como por usuario concreto del sistema. Para esta prueba vamos a utilizar dos cuentas: una denominada `citic`, que no está limitada, y otra llamada `guest`, que

sólo puede usar como máximo 8 CPUs. Cuando ejecutamos el mismo trabajo con ambas (Figura 4.11) vemos que en la cola una de las peticiones no se llega a cumplir por políticas del *accounting*.

```
worker@loginslurmtest:/datasharing/jobs/benchmarks$ sbatch --account=guest --partition=compute benchmark_slurm.job
Submitted batch job 76
worker@loginslurmtest:/datasharing/jobs/benchmarks$ squeue
  JOBID PARTITION  NAME      USER ST       TIME  NODES NODELIST(REASON)
    76  compute_benchmark  worker PD      0:00      1  (AssocGrpCpuLimit)
worker@loginslurmtest:/datasharing/jobs/benchmarks$ 
worker@loginslurmtest:/datasharing/jobs/benchmarks$ 
worker@loginslurmtest:/datasharing/jobs/benchmarks$ 
worker@loginslurmtest:/datasharing/jobs/benchmarks$ squeue
  JOBID PARTITION  NAME      USER ST       TIME  NODES NODELIST(REASON)
    76  compute_benchmark  worker PD      0:00      1  (AssocGrpCpuLimit)
worker@loginslurmtest:/datasharing/jobs/benchmarks$ 
worker@loginslurmtest:/datasharing/jobs/benchmarks$ 
worker@loginslurmtest:/datasharing/jobs/benchmarks$ sbatch --account=citic --partition=compute benchmark_slurm.job
Submitted batch job 77
worker@loginslurmtest:/datasharing/jobs/benchmarks$ squeue
  JOBID PARTITION  NAME      USER ST       TIME  NODES NODELIST(REASON)
    76  compute_benchmark  worker PD      0:00      1  (AssocGrpCpuLimit)
    77  compute_benchmark  worker R      0:01      1  compilslurmtest
worker@loginslurmtest:/datasharing/jobs/benchmarks$ 
worker@loginslurmtest:/datasharing/jobs/benchmarks$ 
```

Figura 4.11: Salida de la prueba de restricción de recursos a nivel de cuenta

Esto se debe a que la base de datos controla los permisos de cada cuenta registrada por medio de una tabla de asociaciones (Figura 4.12), donde las columnas representan los parámetros a considerar antes de confirmar el procesamiento de una solicitud, desde el máximo de trabajos que puede ejecutar hasta la prioridad de sus ejecuciones o la fracción de sistema (*fairshare*) que tienen disponible.

Todas estas variables se pueden configurar utilizando el comando `sacctmgr modify account <cuenta> set <propiedad>=<recurso/valor>`

```
maestro@maestroslurmtest:~$ sacctmgr show associations
  Cluster Account      User Partition Share Priority GrpJobs   GrpTRES GrpSubmit   GrpWall  GrpTRESMins MaxJobs
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
  citic-slur  root          root        1
  citic-slur  root          root        1
  citic-slur  citic         citic        1
  citic-slur  citic         worker       1
  citic-slur  guest         guest        1
  citic-slur  guest         worker       1
cpu=8
maestro@maestroslurmtest:~$ 
```

Figura 4.12: Fragmento de la tabla de asociaciones recursos-cuenta en SLURM

4.5.6 Monitorización de las colas

Si bien las herramientas que ofrece **SLURM** son bastante útiles a la hora de revisar el estado de las máquinas y peticiones, estén o no en curso, el hecho de tener que comprobar estos aspectos manualmente no deja de ser una tarea algo tediosa. Por ende, se decidió integrar un cuadro de mando con interfaz web para automatizar este proceso de recolección y permitir una supervisión continua de los elementos clave. Para llevar a cabo este montaje, es preciso realizar los siguientes pasos:

1. Desplegar en el nodo maestro un entorno de Prometheus, quien actuará de intermediario entre el *software* de visualización y las máquinas del clúster.

- (a) Crear un usuario con el mismo nombre junto a los directorios `/etc/prometheus` y `/var/lib/prometheus`, de los cuales será su propietario.
 - (b) Tras descargar y descomprimir el paquete asociado, copiar los archivos `prometheus` y `promtool` a la ruta donde se alojan los ejecutables a nivel de usuario (`/usr/local/bin`). También indicar en el fichero del servicio que utilice los directorios del punto anterior para la configuración y el almacenamiento de las series temporales.
 - (c) Crear el fichero `prometheus.yml` en `/etc/prometheus` y modificarlo para que incluya los `targets` adecuados, i.e. las máquinas a supervisar, así como los puertos sobre los que se establecerá la comunicación (ver Anexo A.2).
2. Instalar en los nodos correspondientes los servicios de recopilación o exportadores que transmitirán los datos obtenidos al servicio de Prometheus.
 3. Instalar el paquete de Grafana en el nodo maestro, acceder al apartado de ajustes en la interfaz web del aplicativo y agregar la dirección URL de Prometheus como fuente de datos (ver figura 4.13).

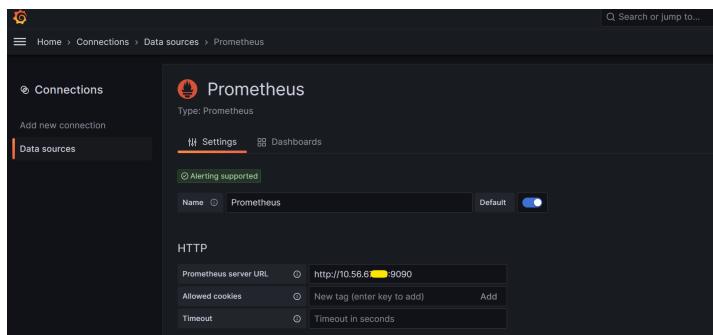


Figura 4.13: Agregación de fuente de datos a Grafana

Tras esto ya podremos crear y personalizar gráficos de diversos estilos (series temporales, histogramas, diagramas de barras, etc.) a partir de consultas sobre las variables extraídas por los exportadores. Cabe señalar que es posible agruparlos por filas desplegables y modificar el intervalo temporal sobre el que se visualiza dicho contenido, siendo por defecto cada 6 horas. En las figuras 4.14, 4.15, 4.16 y 4.17 se distinguen los principales bloques del panel elaborado.

Estado de las particiones del clúster

En este fragmento se incluye la evolución de la temperatura en los procesadores, la cantidad de memoria RAM disponible y cómo se han ido asignando ambos componentes en cada instante.

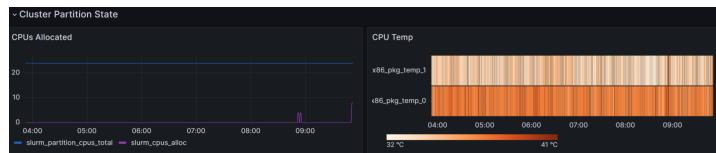


Figura 4.14: Asignación de núcleos y temperatura de procesadores en partición de cómputo

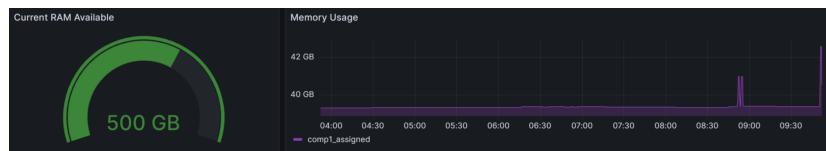


Figura 4.15: Disponibilidad y asignación de memoria RAM en partición de cómputo

Registro de trabajos

Esta parte mantiene información sobre el estado de la cola y las ejecuciones finalizadas, cuántos núcleos de CPU y solicitudes se han realizado en cada cuenta y un contador de aquellas peticiones que hayan sido canceladas, prorrogadas o que han sufrido algún fallo desde que se puso en marcha por última vez la infraestructura.

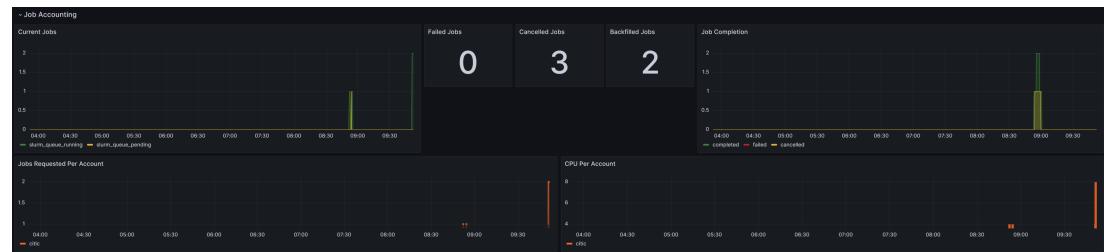


Figura 4.16: Exposición de metadatos sobre ejecuciones procesadas

Estado de las tarjetas gráficas

Finalmente tenemos el bloque sobre las GPUs, donde se muestra la temperatura y la memoria que está siendo utilizada en ese instante, además del porcentaje de utilización y el consumo de energía en watos, dado que pueden ayudarnos a determinar si hay algún uso anómalo o si existen problemas en el suministro eléctrico.

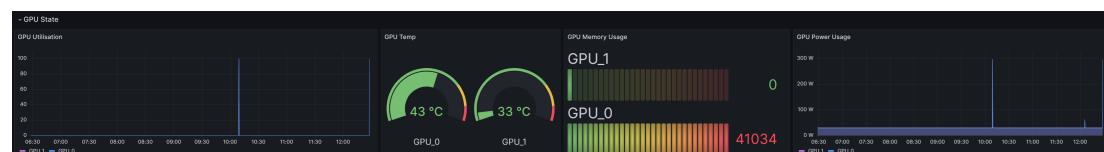


Figura 4.17: Visualización del estado de las GPUs en partición de cómputo

Capítulo 5

Configuración de virtualizadores

Una vez visto el proceso de creación y puesta en marcha del sistema de colas, pasamos a la parte de las plataformas de virtualización. En este caso daremos una serie de indicaciones sobre cómo se ha elaborado cada una y qué adaptaciones han sido necesarias para poder realizar las mismas pruebas de rendimiento que en el capítulo anterior.

5.1 Despliegue en Hyper-V

En primer lugar, es necesario que el servidor tenga instalada una instancia de Windows Server, habiendo recurrido en este caso a la versión estándar de 2019. Tras esto debemos asegurarnos de que la interfaz de red se encuentre operativa, ya que de lo contrario no podremos incorporar los módulos correspondientes al hipervisor. Para ello entramos en la configuración del adaptador (figura 5.1) y le asignamos de forma estática una dirección IP, además de incluir la puerta de enlace y las entradas de los servidores DNS. Para agilizar este proceso hemos tomado una dirección del segmento que fue utilizado como red de gestión en el clúster de SLURM.

Cabe mencionar que admite tanto interfaz gráfica como [CLI](#) para la gestión de los elementos del entorno, ya sean componentes o dispositivos.

5.1.1 Incorporación del rol

Con el servidor ya preparado, accedemos al panel de administración, seleccionamos la opción **Administrar > Agregar roles y características** y en el apartado de roles del servidor dentro del asistente marcamos la casilla de **Hyper-V** (figura 5.2). Posteriormente se nos dirá de establecer algunos ajustes acerca de los conmutadores de red y las rutas en las que se almacenan los metadatos de las máquinas creadas, aunque como veremos más adelante estos se pueden modificar en cualquier momento. Por último se mostrará una pestaña a modo

de resumen para confirmar los cambios, solicitando un reinicio del servidor al completar la instalación.

Ahora si introducimos en la barra de búsqueda “Administrador de Hyper-V” se abrirá una vista como la de la figura 5.3 con los servidores incluidos en la agrupación (pese a que tengamos solamente uno en esta situación), las máquinas que contienen y las acciones que se pueden realizar sobre los recursos.



Figura 5.3: Panel principal de Hyper-V

5.1.2 Acceso a red

El siguiente paso es convertir el adaptador de red en un commutador virtual, ya que precisamos que las máquinas virtuales tengan conectividad a Internet. Para ello en **Acciones > Administrador de commutadores virtuales** seleccionamos la opción de crear uno nuevo, y debemos cerciorarnos que sea externo. También es esencial marcar la casilla de compartición del adaptador (figura 5.4), ya que de esta manera evitamos perder la conexión remota con el servidor y habilitamos una conexión de puente para poder designarle a las máquinas una IP del mismo segmento que el equipo anfitrión.

Por otro lado, Windows deshabilita por defecto la recepción de mensajes ICMP, lo que dificulta la realización de pruebas de conexión y/o medición de la tasa de bits entre este servidor y otra máquina de su entorno. No obstante, podemos modificar este comportamiento directamente desde el **firewall** del sistema. Simplemente en la sección de **Reglas de entrada** buscamos la correspondiente a esta clase y seleccionamos **Acciones > Habilitar regla** (ver figura 5.5).

5.1.3 Creación de máquina virtual

Finalmente vamos a ver el proceso para generar una máquina de prueba. En este caso accedemos a **Acciones > Nuevo > Máquina Virtual**, y nos aparecerá un asistente en el que podemos directamente dejarla con los parámetros por defecto o bien definir la asignación de los recursos, el método de instalación y otros aspectos clave. Uno de ellos es la generación

Capítulo 5. Configuración de virtualizadores

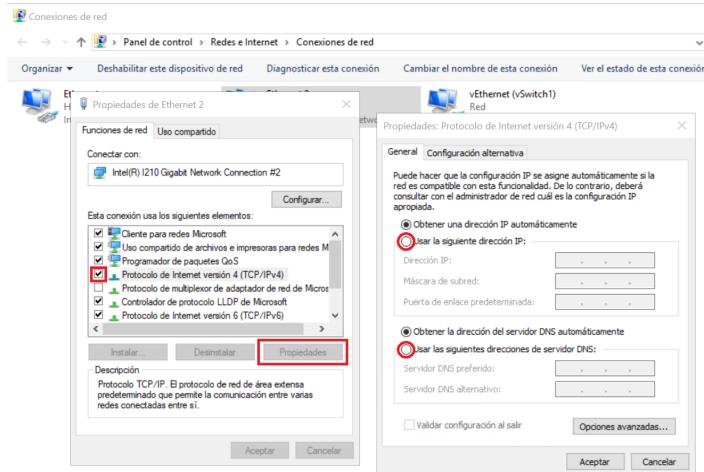


Figura 5.1: Configuración estática del adaptador de red en Windows

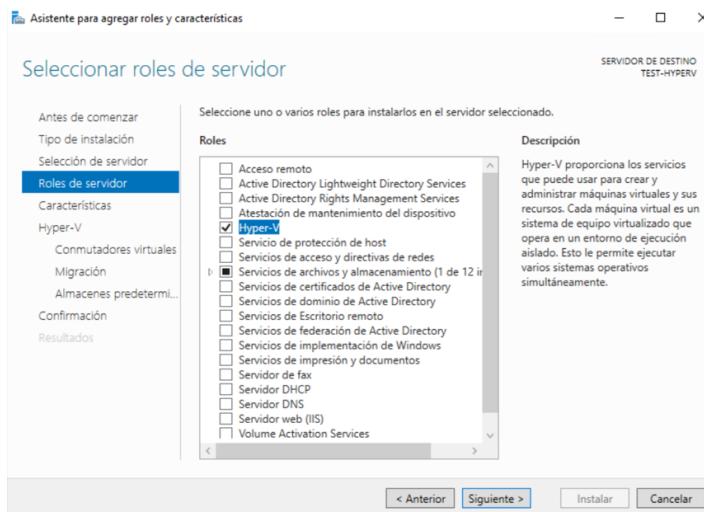


Figura 5.2: Instalación del rol de Hyper-V

5.1. Despliegue en Hyper-V

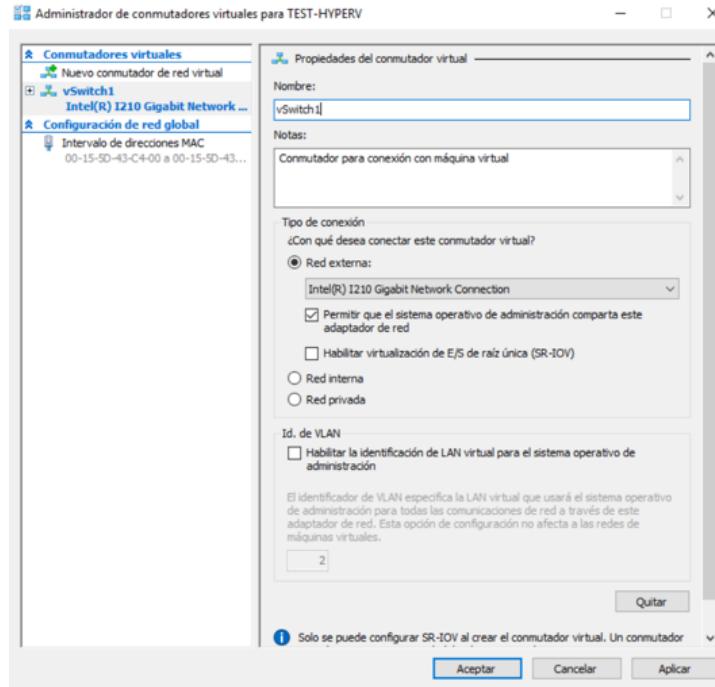


Figura 5.4: Creación del vSwitch en Hyper-V

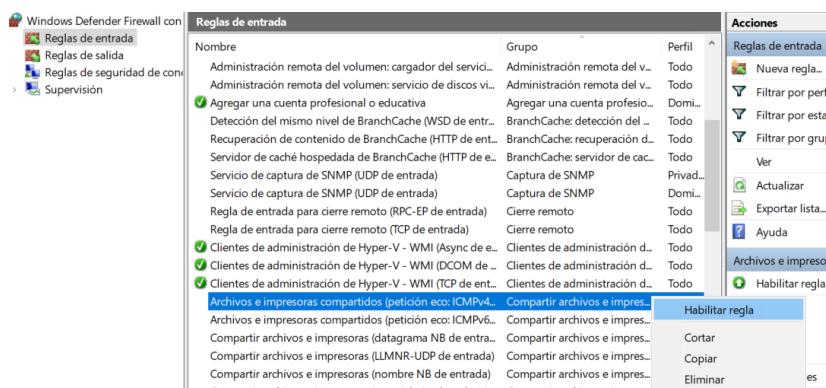


Figura 5.5: Activación de protocolo ICMP

del dispositivo, la cual dependerá del tipo de arquitectura que vayamos a emular: si contamos con un sistema antiguo (i.e que utilice BIOS en vez de UEFI o que sea de 32 bits) entonces es preferible indicarle la primera; de lo contrario podremos asignarle cualquiera de las dos sin ningún problema, aunque con la segunda contaremos con más opciones para las tareas de gestión. En la figura 5.6 se puede visualizar la pantalla final del asistente con los puntos principales.

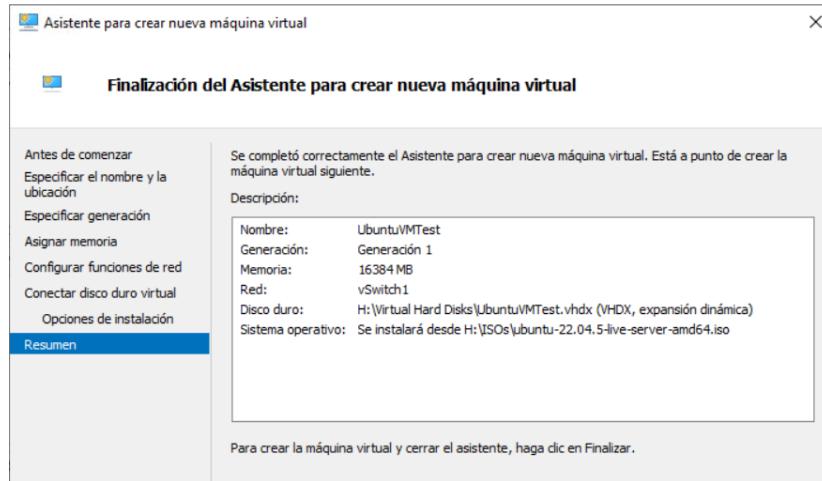


Figura 5.6: Asistente de máquina virtual en Hyper-V

5.2 Despliegue en Proxmox

Disponemos de dos formas de llevar a cabo esta implementación:

1. Descargar una **ISO propia** de Proxmox y utilizar su instalador. Reduce la complejidad del proceso al automatizar algunas configuraciones iniciales, garantiza soporte y está optimizado para las labores de hipervisor, pero no permite personalizar el esquema de particiones del almacenamiento y modificar ciertos aspectos del sistema se vuelve más complejo.
2. Utilizar una distribución de **Debian** como base, puesto que Proxmox es en definitiva una capa de abstracción de este, e instalar por encima los paquetes y dependencias necesarias para su despliegue. Es más tedioso pero permite un mayor control sobre el entorno.

En una rama de producción lo más simple es recurrir a la imagen del hipervisor, pero dado que estamos gestionando un entorno de prueba y queremos contar con la mayor libertad posible, hemos decidido utilizar la segunda opción.

5.2.1 Instalación de dependencias

Lo primero es agregar en la lista de repositorios aquel correspondiente al *kernel* de PVE (*Proxmox Virtual Environment*). Al no contar con una licencia profesional para su despliegue, utilizaremos la vertiente sin suscripción en vez de la *enterprise*.

```

1 echo "deb http://download.proxmox.com/debian/pve bookworm
      pve-no-subscription" > /etc/apt/sources.list.d/pve-install-repo.list
2
3 # Añade la clave del repositorio
4 wget https://enterprise.proxmox.com/debian/proxmox-release-bookworm.gpg
      -O /etc/apt/trusted.gpg.d/proxmox-release.gpg
5
6 # Actualizar la lista e instalar los paquetes
7 apt update
8 apt install proxmox-ve postfix open-iscsi

```

Es probable que nos salte una pestaña para configurar la parte de *postfix*, sobre lo cual escogemos la opción de **Sitio de Internet** (figura 5.7). No obstante también sería igual de válido dejarlo como sólo local, ya que únicamente afecta al envío de alertas por correo de forma externa.

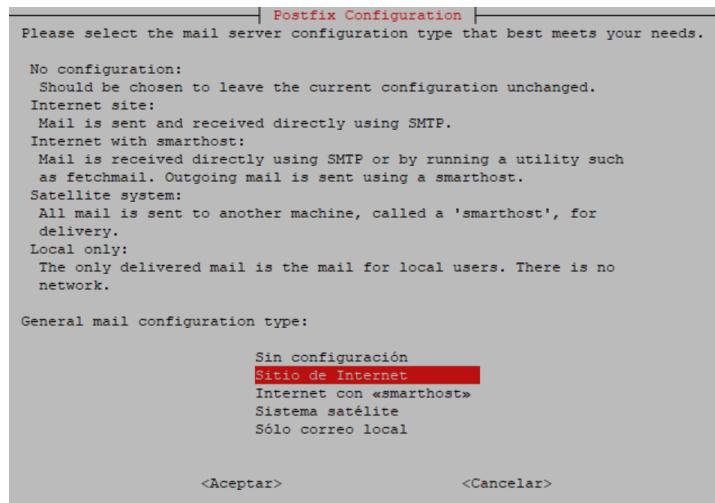


Figura 5.7: Configuración de postfix

Una vez reiniciado el servidor podremos acceder a la interfaz web a través del puerto 8006, y al entrar nos pedirá las credenciales de un usuario administrador (figura 5.8). Por defecto este rol lo ocupa el usuario *root* del sistema, pero se puede llegar a asignar otro perfil en su lugar.

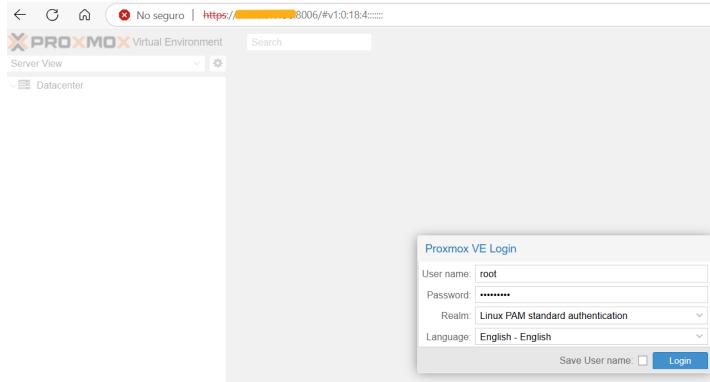


Figura 5.8: Pantalla de login de Proxmox

5.2.2 Configuración de interfaz puente

De forma similar a lo que sucede en Hyper-V, debemos crear un adaptador virtual para otorgar conectividad externa a las máquinas. Para ello es necesario entrar en la sección **Network** del servidor, escoger la interfaz que está siendo utilizada actualmente por el servidor y copiar su configuración en una nueva entrada de tipo **Linux Bridge**, asegurándonos además de que la interfaz vieja no se inicie luego de forma automática. En las figuras 5.9 y 5.10 se pueden apreciar los pasos realizados. Al pulsar en **Apply Configuration**, provocaremos que se reinicie el servicio de red y se levanten automáticamente aquellas interfaces que así se hayan solicitado.

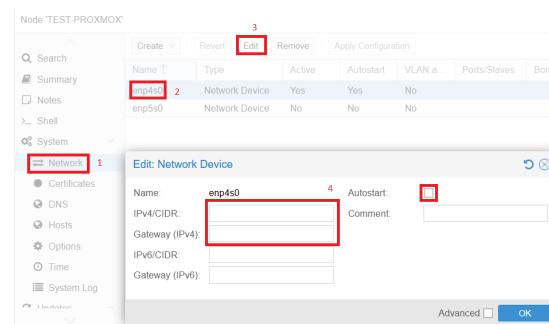


Figura 5.9: Desactivación de interfaz de red física en Proxmox

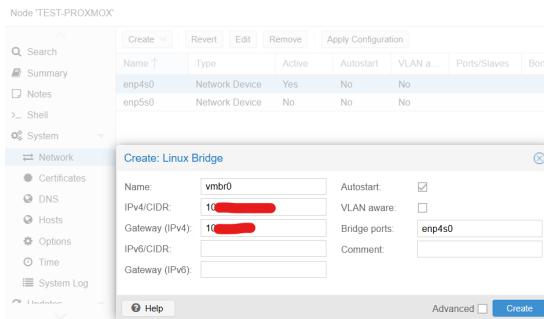


Figura 5.10: Creación de adaptador puente en Proxmox

Este proceso también se puede llevar a cabo con las siguientes modificaciones sobre el fichero `/etc/network/interfaces`. De hecho, lo que se está haciendo por debajo en el método web es delegar dichos cambios en el archivo, puesto que cuando realizamos cualquier ajuste aparece en la parte inferior de la pantalla una vista previa con los comandos a ejecutar.

```

1 auto lo
2
3   iface lo inet loopback
4
5   iface <interfaz> inet manual
6
7     dns-nameservers <lista_servidores>
8
9 auto vmbr0
10
11 iface vmbr0 inet static
12
13   address <IP_servidor>/<máscara_red>
14
15   gateway <IP_gateway>
16
17   bridge-ports <interfaz>
18
19   bridge-stp off    # No activamos spanning tree, i.e el puente
20   se levanta de forma inmediata
21
22   bridge-fd 0        # Eliminamos retraso en el reenvío de tráfico
23   tras un cambio en la red

```

5.2.3 Creación de máquina virtual

Antes de poder generar la máquina de prueba debemos subir al entorno de Proxmox la imagen ISO del sistema invitado, puesto que el horizonte conocido del asistente abarca a partir

del punto de montaje del entorno. En consecuencia se vuelve necesario transferir el archivo desde su ruta de origen hasta el almacenamiento del hipervisor (figura 5.11).



```
Task viewer: Copy data
Output Status
Stop Download
starting file import from: /var/tmp/pveupload-5a2cf20bd85ecb78036b756684fc02c6
target node: TEST-PROXMOX
target file: /var/lib/vz/template/iso/ubuntu-22.04.5-live-server-amd64.iso
file size is: 2136926208
command: cp -r /var/tmp/pveupload-5a2cf20bd85ecb78036b756684fc02c6 /var/lib/vz/template/iso/ubuntu-22.04.5-live-server-amd64.iso
finished file import successfully
TASK OK
```

Figura 5.11: Subida de una imagen ISO a Proxmox

Tras esto sólo nos queda definir las características de la máquina, siendo especialmente relevantes el tipo de bios (OVMF al tratarse de un sistema UEFI) y el formato del disco duro virtual. En la figura 5.12 se incluyen todas las especificaciones asignadas. Recalcar además que cuando se crea un nuevo equipo el sistema le determina un identificador numérico, con la finalidad de facilitar la automatización de ciertos procesos y evitar así ambigüedades en los nombres.

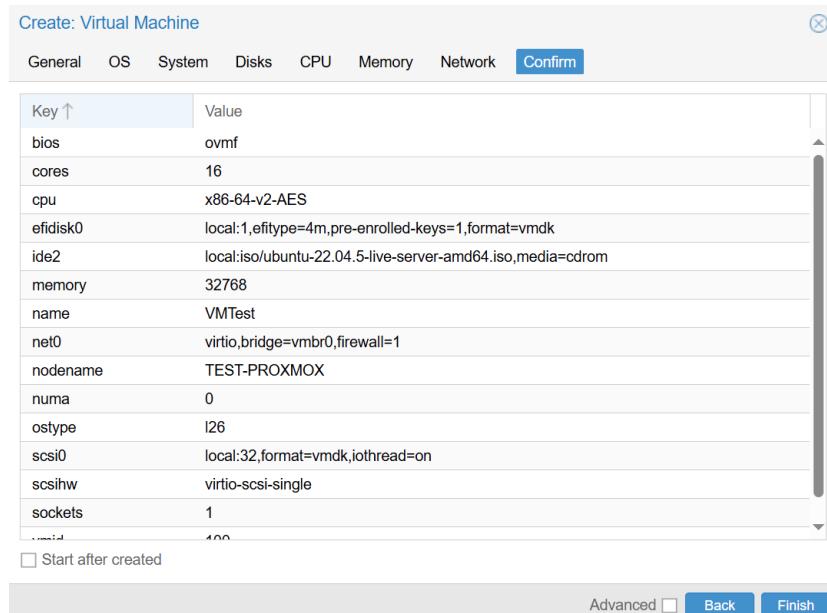


Figura 5.12: Asistente de máquina virtual en Proxmox

5.3 Despliegue en VMware ESXi

Actualmente no resulta posible adquirir una imagen nueva de instalación análoga a las que se utilizan en los servidores del *citic*, por no hablar de las licencias de uso. Afortunadamente pudimos reutilizar una imagen ISO cedida por la organización, por lo que expondremos directamente los pasos a seguir para desplegar la máquina virtual en este entorno.

5.3.1 Definición de almacén de datos

A diferencia de las otras dos plataformas, es necesario designar una partición propia o un medio de almacenamiento remoto para alojar los datos de las máquinas creadas, así como poder importar los archivos ISO de los sistemas operativos invitados. En este caso hemos optado por la segunda opción, aprovechando el sistema NFS que habíamos montado para el clúster de **SLURM**. Para ello en la interfaz web del hipervisor, en la sección de **Almacenamiento > Almacenes de datos** seleccionamos **Nuevo almacén de datos**. Ya dentro del asistente escogemos montar un NFS remoto y la ruta de la carpeta a compartir. Al finalizar el proceso podremos acceder a la nueva entrada y ver su contenido (figura 5.13), así como agregar nuevos ficheros y directorios.

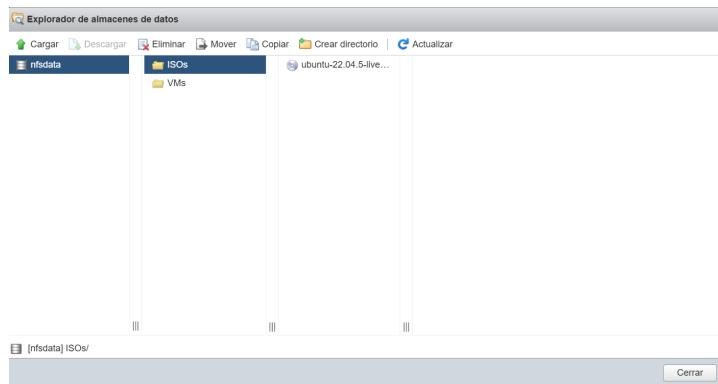


Figura 5.13: Montaje de NFS en ESXi

5.3.2 Asociación de grupo de puertos

Otro aspecto que cambia ligeramente es el tráfico de red. Para disponer de conectividad se debe crear un adaptador VMKernel para exponer cada una de las interfaces del *host*, y a su vez configurar un conmutador virtual que se comunique con dichos puertos. A esto se le suma el concepto de grupo de puertos, un método para mantener una agregación a nivel lógico entre las conexiones de una misma VLAN, y que deben asignarse a las máquinas durante su creación. Al no haber etiquetado ninguno de los adaptadores para este fin, debemos configurar

un grupo de puertos con ID de VLAN 0 (i.e. que admita tráfico dirigido a cualquier segmento). En la figura 5.14 se muestra la topología resultante.

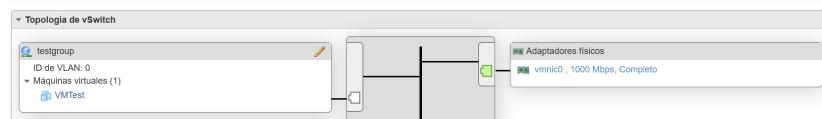


Figura 5.14: Topología del grupo de puertos en ESXi

5.3.3 Creación de máquina virtual

Finalmente debemos acceder a la sección de *Máquinas virtuales* y seleccionar **Crear/Registrar máquina virtual**. El asistente nos pedirá seleccionar la familia del sistema operativo y la compatibilidad con respecto a la versión del hipervisor (seleccionamos la más reciente). Posteriormente se ha de indicar el almacenamiento de datos a utilizar, además de personalizar las características de los diferentes componentes, tales como el modo de aprovisionamiento del disco virtual o la adición de controladores periféricos. Es crucial asegurarse de que en el desplegable de **Opciones de máquina > Opciones de arranque** la entrada de *firmware* sea de tipo UEFI, ya que de lo contrario no se podrán aplicar ciertas funcionalidades sobre la máquina (algo similar a lo que sucede en Hyper-V) entre ellas la transferencia directa de *GPU*. La configuración final aparece reflejada en la figura 5.15.

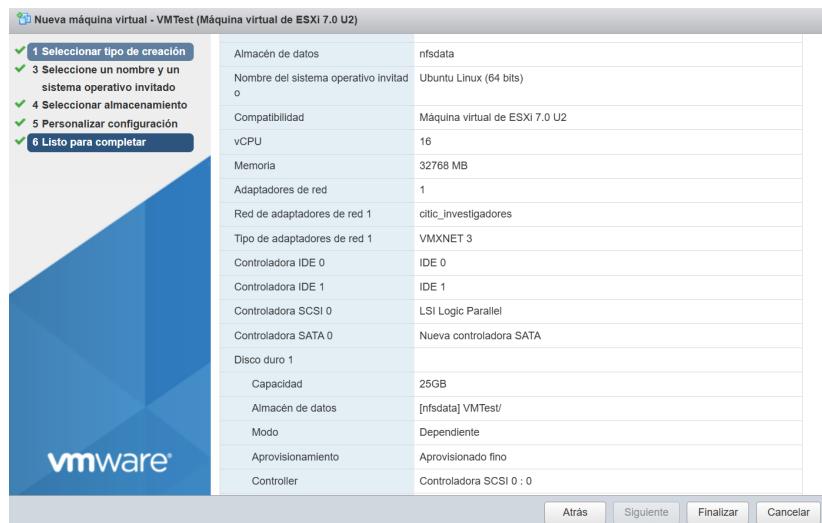


Figura 5.15: Asistente de máquina virtual en ESXi

5.3.4 Conexión de tarjetas gráficas por passthrough

De cara a tener una referencia sobre la que comparar las pruebas a nivel de GPU, también hemos integrado la transferencia de las tarjetas gráficas del servidor a la máquina virtual. En VMware hay dos formas de llevar a cabo este proceso: por troceado de memoria, que consiste en fragmentar el dispositivo físico en tarjetas virtuales o vGPUs de diferentes perfiles y tamaños, y por transferencia directa (*passthrough*), en la que asigna la totalidad de su espacio. Por sencillez, y puesto que la opción de particionado requiere de una licencia adicional, hemos recurrido al segundo método.

Para poder habilitarlo es necesario acceder a la sección de **Hardware > PCI Devices** del servidor, marcar aquellas entradas relativas a las GPU y seleccionar **Alternar Acceso Directo** (Figura 5.16). Posteriormente, con la máquina virtual apagada entramos en su apartado de ajustes, le agregamos dos dispositivos de tipo PCI y reservamos por completo la memoria del sistema invitado (Figura 5.17). Asimismo, en caso de que diese algún error durante la secuencia de encendido será necesario activar en la BIOS la opción "Above 4G Encoding" e incluir en el apartado de configuración avanzada de la máquina virtual las siguientes variables de entorno:

```

1 pcihole.start = 2048
2 pciPassthru.use64bitMMIO = TRUE
3 pciPassthru.use64bitMMIOSizeGB = 256
4 # Se recomienda utilizar para el size una potencia de dos superior
   a la suma de la memoria ocupada por las GPU.

```

Esto se debe a que algunas tarjetas gráficas, especialmente las más modernas, precisan de un espacio de direcciones mayor para realizar el mapeado de memoria y que puedan ser reconocidas de forma correcta por el sistema operativo. Por último quedaría arrancar de nuevo la máquina virtual e instalar la versión de los controladores correspondiente.

Aclarar además que el motivo de no llevar a cabo este proceso en el resto de hipervisores reside en que bien su configuración actual no permite aplicar dicho ajuste, precisando una edición superior de pago, o bien pese a seguir la documentación oficial no ha sido posible efectuarlo con éxito.

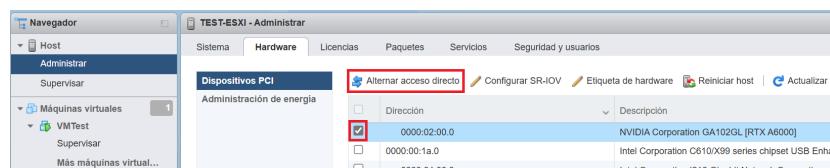


Figura 5.16: Activación de passthrough sobre las tarjetas gráficas

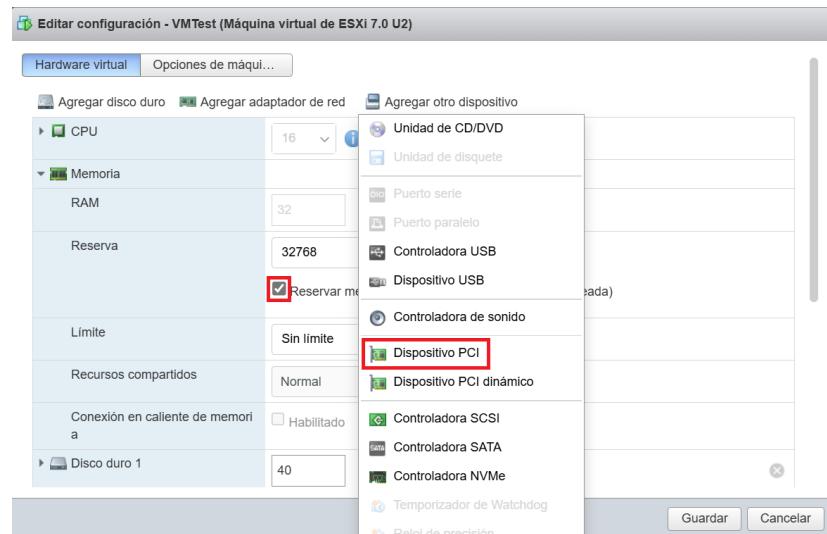


Figura 5.17: Designación de dispositivos PCI

Capítulo 6

Resultados obtenidos

TRAS haber expuesto todos los despliegues, solamente nos queda por visualizar y comparar las métricas obtenidas con las pruebas de rendimiento. En este apartado comentaremos en detalle los parámetros de las simulaciones llevadas a cabo y trataremos de aportar una justificación sobre lo recopilado en las mismas. A mayores incluiremos una breve descripción acerca de las políticas en sus tarifas

6.1 Prueba de rendimiento

Tal y como se dijo anteriormente, la ejecución utilizada como referente principal para este contraste es el fichero de *compbench* mencionado en la parte de [SLURM](#). Este se encuentra dividido en varios bloques:

- Simulación con *sysbench* de cálculos matemáticos a nivel de CPU y transmisión de segmentos de datos por memoria para evaluar el procesamiento de cargas concurrentes.
- Creación mediante *fio* de ficheros temporales sobre los que aplicar múltiples operaciones I/O, ya sean accesos secuenciales o aleatorios.
- Establecimiento de comunicaciones cliente-servidor usando *iperf3* para analizar el estado de la red.

En cuanto a la parametrización, salvo ciertas variables como el número de hilos en las ejecuciones o los accesos aleatorios en disco, se ha recurrido a valores por defecto para ofrecer una visión exploratoria y homogénea de cada una de las plataformas.

Asimismo, para evitar sesgos innecesarios hemos realizado 5 iteraciones consecutivas de dicha ejecución, y se han estandarizado los requisitos que deben cumplir tanto las máquinas virtuales de los hipervisores candidatos como los trabajos ejecutados en [SLURM](#), a saber: 16 núcleos de CPU, 32 GB de memoria RAM y discos SSD SATA REV 3 de 6Gb/s. De igual

forma, se ha tratado de corroborar los resultados con estadísticas recabadas en procedimientos similares [49, 50].

Mencionar también que el objetivo primordial de esta comparativa es determinar aquella plataforma capaz de mantener un mayor compromiso entre todos los componentes, puesto que la alta disponibilidad es uno de los puntos fuertes que ha de conservar un centro de datos de este calibre.

6.1.1 Métricas a considerar

Para establecer una comparación adecuada de los candidatos, es imprescindible tomar valores numéricos de aquellas características que puedan resultar de especial interés. En particular, hemos seleccionado los siguientes parámetros:

- Nº eventos en CPU.
- Latencia de CPU (ms).
- Ancho de banda en memoria (MiB/s).
- Promedio de IOPS, i.e. operaciones de entrada-salida en disco.
- Latencia en lectura de disco (μ s).
- Latencia en escritura de disco (μ s).
- Ancho de banda en red (Mbps).
- Nº retransmisiones en red.

6.1.2 Extracción y análisis de datos

De cara a facilitar la visualización de las métricas, hemos automatizado su búsqueda y posterior volcado a un archivo de Excel a través del código que se muestra en el Anexo A.4. En él se emplean sobre todo expresiones regulares que se adaptan al formato de los campos deseados y sobre las cuales se traspasan los valores correspondientes a variables de un diccionario. Además requiere que se le pase como argumentos la ruta del directorio donde se encuentran los ficheros con los resultados y el nombre del archivo CSV sobre el que plasmar las entradas de cada iteración, quedando una ejecución de la forma:

```
1 python benchtocs.py <directorío_entrada> --output <fichero.csv>
```

Matriz de correlaciones

Un aspecto que se debe tener en cuenta antes de llevar a cabo la comparativa es ver qué atributos presentan una vinculación más fuerte en su comportamiento, ya que de esta forma se tiene una primera impresión de las tendencias en los valores. En la matriz¹ de la figura 6.1 podemos ver que, al margen de las asociaciones entre atributos de un mismo componente, existe una relación significativa de las operaciones en disco con respecto a la cantidad de eventos en CPU y al ancho de banda en red, así como de la tasa de transferencia de memoria en el tiempo de latencia sobre los dispositivos de almacenamiento, ya sea a nivel de lectura o escritura.

Esto implica que cualquier desequilibrio en la red puede alterar el funcionamiento interno de las tareas del servidor y su intercambio de información con las máquinas virtuales. También se deduce que las operaciones de disco de carga elevada suponen una mayor congestión tanto en el procesamiento a nivel de CPU como en memoria, incrementando por ende su valor de retardo.

	CPU event	CPU latency	Memory th	Disk IOPS	Disk Read	Disk Write	Network Bandwidth (Mbps)	Network Retransmits
CPU events/s		-0,99862	-0,57559	-0,94121	0,013828	-0,08306	-0,500500586	0,41083
CPU latency avg(ms)		0,538318	0,953218		-0,04278	0,041938	0,485891028	-0,40305
Memory throughput (Mbps)			0,288377	0,616233	0,822302		0,537228541	-0,35235
Disk IOPS (avg)				-0,26502	-0,23367		0,315450109	-0,5337
Disk Read Latency (avg[us])					0,629394		0,511270089	-0,01796
Disk Write Latency (avg[us])						0,334110484	-0,16069	
Network Bandwidth (Mbps)							0,281844	
Network Retransmits								

Figura 6.1: Matriz de correlaciones entre métricas principales

Contraste gráfico

Eventos de CPU

En la gráfica de la figura 6.2 tenemos un diagrama de barras con el promedio de eventos por segundo que es capaz de realizar concurrentemente los procesadores de las máquinas virtuales en cada plataforma. EsXi y Proxmox presentan valores prácticamente idénticos, mientras que Hyper-v es la que más sufre en este apartado. Estas variaciones pueden deberse a la forma en la que se optimicen las llamadas al sistema entre la máquina virtual y el hipervisor, así como a la carga adicional destinada a las tareas de administración de los recursos y procesos de virtualización.

En el caso de [SLURM](#), al ser una implementación sobre componentes físicos es capaz de aprovechar el rendimiento puro de los componentes, algo que también se notará en los siguientes puntos.

¹ Sólo se muestra la mitad superior dado que mantiene propiedad de simetría.

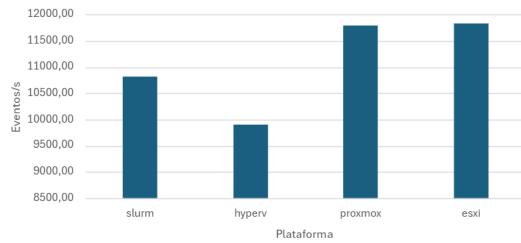


Figura 6.2: Promedio del número de eventos en CPU

Transmisión en memoria

En este caso la figura 6.3 muestra que el entorno de [SLURM](#) es el que aporta mejores resultados, ya que al no haber una capa de virtualización intermedia no precisa traducir direcciones adicionales de memoria. Por otro lado, Hyper-V presenta una diferencia razonable con respecto a los otros hipervisores, lo cual denota que el sistema operativo de Windows implementa un mecanismo más eficiente para emular la memoria que en distribuciones UNIX.

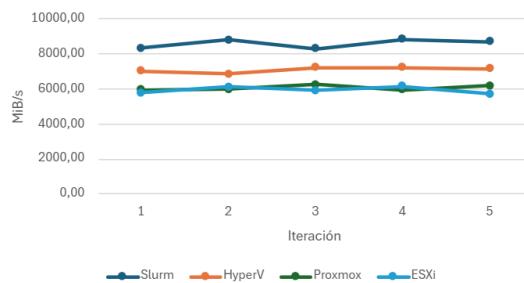


Figura 6.3: Evolución del ancho de banda en memoria por iteración

Relación eventos/latencia (CPU)

Extrapolando el razonamiento de la primera gráfica se puede llegar a deducir los valores de la figura 6.4, en donde Proxmox y EsXi vuelven a ser los referentes (a menor tiempo de espera mayor es el número de eventos a albergar en el mismo intervalo).

Relación eventos/latencia (Disco)

Las figuras 6.5 y 6.6 son sin duda de las más interesantes en esta comparativa. En ellas se puede apreciar una cierta dispersión en los valores de [SLURM](#) (siendo más notable en la parte de lectura), motivado en gran medida por los retardos derivados del protocolo NFS y la carga que suponga la transmisión de las peticiones por red, lo que a su vez justificaría la correlación entre estas dos variables.

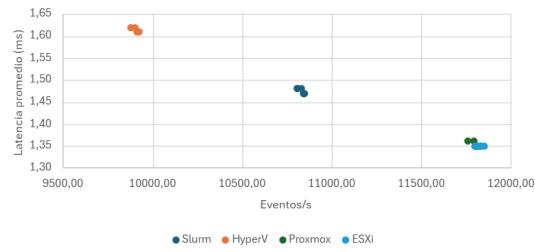


Figura 6.4: Correspondencia entre eventos de CPU y su latencia promedio

Asimismo, se observa que Hyper-V es la que puede llevar a cabo más operaciones de entrada-salida, con una latencia prácticamente nula, y que tanto EsXi como Proxmox se ven especialmente mermados en el número de IOPS, posiblemente influenciado por cómo se produce el formateo y/o segmentación de los bloques de datos en los discos virtuales (vmdk y qcow2 respectivamente), que si bien permiten un uso del espacio más provechoso pueden acarrear consigo diversas penalizaciones a la hora de recuperar la información e incluso problemas de fragmentación o replicado del contenido.

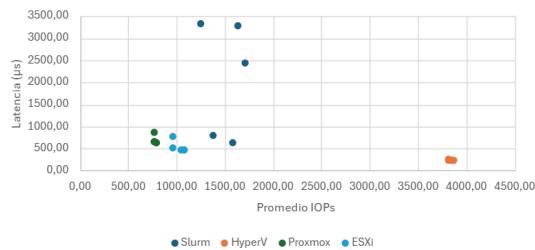


Figura 6.5: Correspondencia entre operaciones I/O de lectura y su latencia promedio

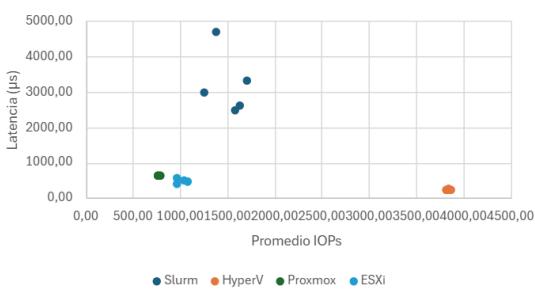


Figura 6.6: Correspondencia entre operaciones I/O de escritura y su latencia promedio

Estado del tráfico de red

Respecto a las pruebas cliente-servidor de *iperf3*, en las figuras 6.7, 6.8 y 6.9 se muestran los valores de cada plataforma en relación con la tasa de bits, la cantidad de paquetes que fue

necesario retransmitir y la ventana de congestión de la red (i.e. cuántos datos consecutivos se pueden enviar antes de tener que confirmar su recepción). Con ello podemos extraer las siguientes conclusiones:

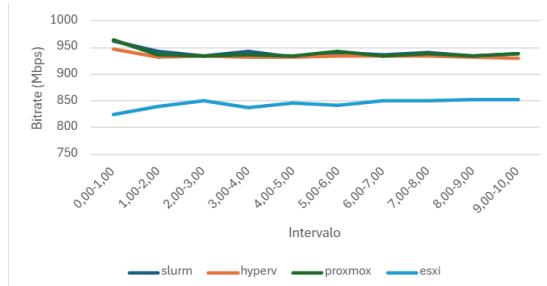


Figura 6.7: Evolución temporal de la tasa de bits en red

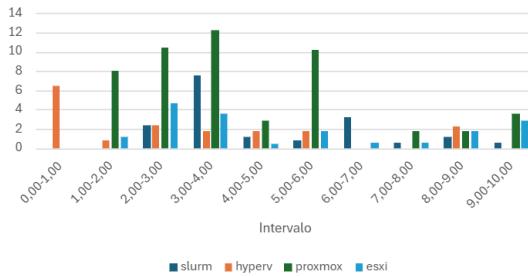


Figura 6.8: Promedio del número de retransmisiones de paquetes por intervalo

- **SLURM** es la que presenta una mejor relación entre capacidad de transferencia y estabilidad en las comunicaciones, dado que al no estar virtualizado las interfaces de red presentan una conexión directa al commutador.
- **EsXi** gestiona la red de forma bastante eficiente y apenas acusa pérdida de paquetes, pero en cuanto a transferencia es ligeramente inferior con respecto al resto de candidatos. Sin embargo, el hecho de poder segmentar el tráfico utilizando grupos de puertos y VLANs dan pie a un potencial de mejora significativo.
- **Proxmox** es claramente la opción más intestable, ya que pese a contar con valores elevados en la tasa de bits y la ventana de congestión, la alta disparidad de esta última, sumado a la gran cantidad de retransmisiones, aporta pocos indicios de fiabilidad. Esto podría estar asociado a la forma en la que el sistema operativo del hipervisor delega el control de las interfaces virtuales.
- **Hyper-V** se mantiene como una solución intermedia, con una gran consistencia en todos los aspectos.

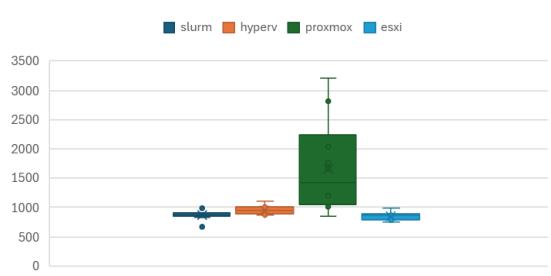


Figura 6.9: Diagrama de caja sobre la variabilidad de la ventana de congestión

6.2 Prueba de Tensorflow

Para esta comparativa usaremos los valores de tiempo, precisión (*accuracy*) y penalización de las predicciones sobre el modelo entrenado en el código de *tf_train* (ver Anexo A.3). Únicamente se tendrán en cuenta para el contraste la plataforma de ESXi y el sistema de colas, siguiendo el mismo razonamiento del apartado 5.3.4, algo que por ende también afecta a la sección venidera.

De las figuras 6.10, 6.11 y 6.12 podemos concluir que la capa de virtualización no afecta en gran medida al rendimiento computacional. Sin embargo, como cabe esperar las ejecuciones sobre dispositivos físicos son bastante más ágiles, llegando incluso a una mejora de prácticamente la mitad.

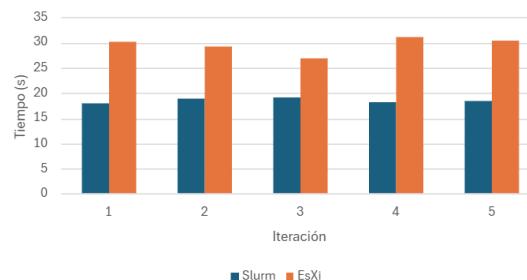


Figura 6.10: Tiempos de ejecución del modelo de Tensorflow por iteración

6.3 Prueba de estrés de GPU

En este apartado incluimos los datos recopilados en cuanto al número de operaciones flotantes y el ritmo de aumento de la temperatura durante la simulación del programa de *gpu-burn* dentro del clúster de SLURM y en una máquina virtual del hipervisor de VMware. En ambos casos se encuentra muestreado en porcentajes relativamente equidistantes de compleción.

En la figura 6.13 se aprecia que el trabajo no se ejecuta de forma simultánea en las dos tarjetas gráficas, lo cual puede estar causado por algún retardo en la asignación de los recursos por parte del sistema, y conforme se acerca al final se experimenta en todos los casos una ligera caída del rendimiento. Pese a ello, la similitud en las tendencias de dicha gráfica y la mostrada en la figura 6.14 sugiere que se ha explotado correctamente la capacidad de las GPU y que no existe ningún problema de refrigeración.

6.4 Desglose económico

Aún habiendo comparado la capacidad técnica de las diferentes plataformas, en un entorno real también hay que tener en cuenta los costes de licenciamiento de los equipos y servicios involucrados, especialmente cuando se tiene una infraestructura a gran escala. Para esta sección y la siguiente vamos a tener en consideración únicamente a los hipervisores, puesto que SLURM no formaría parte de este grupo y, tal y como se comentó en los objetivos, se considera ajeno a este análisis.

En los últimos años multitud de proveedores en este sector han optado por modelos de suscripción, siendo la cantidad de procesadores (o en su defecto núcleos de un procesador) la vara de medir más destacada para la facturación. Asimismo, cada plataforma ofrece a su vez varios planes con diferentes restricciones para abarcar múltiples escenarios. Tras indagar en la documentación de cada hipervisor, se muestran las siguientes ofertas:

Hyper-V incluye el plan estándar (1098.36€/año²) y el plan datacenter (6955.15€/año) [51], siendo que el primero limita a 2 el número de máquinas virtuales que se pueden instanciar en un mismo servidor. Las licencias se emiten en paquetes de al menos 16 núcleos por cada chasis.

Proxmox es a primera vista la plataforma con las propuestas más asequibles, siguiendo con su filosofía de código abierto y accesibilidad a pequeños proyectos. Cada tarifa se obtiene atendiendo al número de procesadores, y actualmente mantiene tres planes: básico, estándar y *premium*, con costes de 305, 530 y 1060€ respectivamente [52].

VMware por su parte es un caso bastante peculiar. Inicialmente permitía la adquisición de licencias perpetuas mediante modelos por procesador y por núcleo, exigiendo un mínimo de 16 núcleos en el primer caso y de 32 en el segundo [53]. Sin embargo, desde la absorción de la empresa por Broadcom en 2024 se han producido numerosas modificaciones en los planes disponibles, pasando a disponer solamente de modelos de suscripción por núcleo, con un mínimo de 16 por procesador (e incluso se estuvo barajando aumentarlo a 72) [54]. Esto complica considerablemente el cálculo de un precio exacto, aunque investigando otras referencias

² En caso de no obtener precio directo en euros, conversión de dólares realizada tomando equivalencias a fecha de mayo de 2025

se pudo deducir un coste por núcleo en la edición *Enterprise plus* (que es la que tienen los servidores del [CITIC](#)) de 150€ [55].

Teniendo en cuenta esta información, suponiendo un escenario con tres servidores del mismo tipo que lo utilizado en las pruebas anteriores, obtendríamos en el primer año unos costes de 14400€ en VMware, 23865.45€ en Hyper-V con edición datacenter y 6360€ en Proxmox con edición *premium*.

6.5 Resumen del procedimiento

Tras haber analizado los datos previos, se ha observado que por lo general las plataformas de virtualización no presentan una pérdida de rendimiento muy elevada con respecto a un sistema sin dicha capa de abstracción, y que en ocasiones llegan a mejorarlo notablemente. Asimismo, se han deducido los siguientes puntos como criterios esenciales de la comparativa:

- En cuanto a CPU y disco, las plataformas tipo UNIX mantienen un comportamiento más estable en el retardo de las operaciones, lo que implica que se pueden adaptar más fácilmente a cambios del entorno.
- A nivel de memoria, la capa de virtualización aporta diferencias sustanciales en la capacidad de intercambio de información, debido al mayor número de ciclos de acceso.
- El tipo de sistema de ficheros y el medio sobre el que sustenta afecta en gran medida al procesamiento de las operaciones de lectura y escritura.
- Los sistemas de Hyper-V y [SLURM](#) son más robustos ante las fluctuaciones de la red, demostrando que el grado de saturación de la misma es directamente proporcional al tamaño de la ventana de congestión y al número de retransmisiones. Asimismo, su eficiencia vendrá marcada por el mecanismo de gestión utilizado y cómo se distribuyan las interfaces asociadas.
- Las aptitudes de las tarjetas gráficas parecen ser independientes al tipo de plataforma considerada, por lo que en última instancia la forma en la que se planifique el uso de cada recurso supondrá el factor diferenciador durante la consecución de las tareas de cómputo.

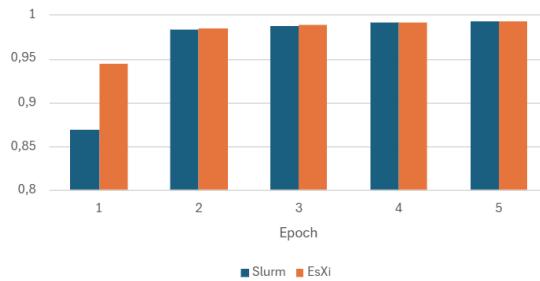


Figura 6.11: Precisión promedio por etapa de las predicciones del modelo de Tensorflow para SLURM y ESXi



Figura 6.12: Penalización promedio por etapa sobre las predicciones de Tensorflow para SLURM y ESXi

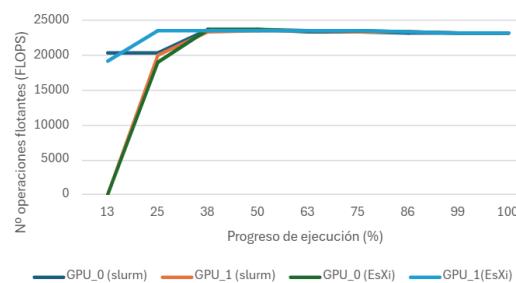


Figura 6.13: Medición progresiva con gpu-burn del rendimiento en FLOPS de las GPU en SLURM y ESXi

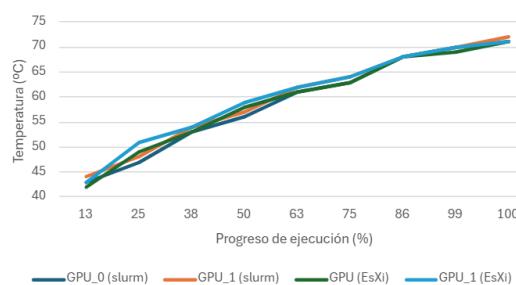


Figura 6.14: Evolución de la temperatura de las GPU en SLURM y ESXi durante ejecución de gpu-burn

Capítulo 7

Conclusiones y trabajo futuro

En este capítulo se expondrán las principales reflexiones extraídas a raíz del desarrollo del proyecto, además de señalar cómo se relaciona su aprendizaje con las diferentes competencias de la titulación y qué aspectos podrían formar parte de una línea de trabajo futura.

7.1 Conclusiones

A lo largo de este trabajo se ha abordado el despliegue de una propuesta de entorno para el alojamiento de las tareas de cómputo del CITIC y el contraste de varias soluciones de virtualización por hipervisor, orientado a una posible migración de los elementos no críticos de su infraestructura.

Con ello en mente, se ha podido exponer de forma detallada los conceptos más relevantes a la hora de reflejar el panorama que envuelve a dicho desarrollo, logrando a su vez una mejor comprensión de los elementos que conforman los centros de datos y los sistemas de gestión de recursos.

Asimismo, el sistema de colas configurado es capaz de planificar cargas de trabajo intensivas, utilizando diversas estrategias para programar su ejecución y controlar tanto el acceso a los recursos como las restricciones que se aplican sobre los mismos a nivel de usuario. De esta forma, se ha llegado a mostrar el comportamiento del clúster con diferentes cargas y su capacidad para manejar múltiples situaciones conflictivas. Por otra parte, su diseño garantiza una distribución coherente de las responsabilidades, siendo capaz de adaptarse cómodamente a los requerimientos del centro y facilitando a largo plazo la completitud de labores de mantenimiento y escalado.

Finalmente, se ha conseguido exhibir las principales diferencias en cuanto al procedimiento de despliegue de cada hipervisor y se ha obtenido una evaluación satisfactoria de su rendimiento a partir de la simulación de varias pruebas de estrés, además de un breve estudio sobre

el coste de cada solución en el primer año.

Con la realización de este proyecto también se ha logrado reforzar algunos conocimientos adquiridos en diversas asignaturas del máster. **Informática como Servicio (ICS)** es quizá la más involucrada, puesto que en ella se mencionan los fundamentos de la computación en la nube y qué mecanismos son útiles a la hora de efectuar despliegues de aplicativos que abstraigan al usuario final. También guardan una estrecha relación:

- **Administración de Sistemas de Información (ASI):** En ella se inculcan las pautas a seguir en la extracción y análisis de requisitos de un desarrollo generalmente vinculado a aplicaciones para el registro y manipulación de datos, aunque son perfectamente aplicables a proyectos de otra índole. Con ello hemos sido capaces de averiguar las inquietudes de los usuarios finales, inferir los casos de uso y plantear ambos en un formato adecuado.
- **Dirección de Proyectos (DP):** Desglosa los principales puntos que afectan a la supervisión de cualquier proyecto, además de aportar buenas prácticas y recomendaciones en el manejo de ciertos apartados, como puede ser la gestión de riesgos.

Del mismo modo, en base a las pruebas llevadas a cabo en los hipervisores se ha visto que no hay una plataforma que destaque ampliamente sobre el resto, sino que cada una presenta distintos puntos fuertes y están pensadas para diferentes escenarios. Si se cuenta con un proyecto de investigación de bajo presupuesto o que no pretende escalar de forma considerable, Proxmox es la alternativa más adecuada; en aquellas infraestructuras que ya se integran herramientas y licencias del ecosistema de Microsoft, o que están fuertemente vinculadas a una implementación de dominios de *Active Directory*, Hyper-V es la mejor solución (pese a la ligera sobrecarga presente en CPU y red). Finalmente, VMware se enfoca en grandes organizaciones con una producción crítica que buscan alta disponibilidad en combinación con un soporte fiable y robusto, como es el caso del [CPD](#) del [CITIC](#).

No obstante, atendiendo exclusivamente al rendimiento puro consideramos que la alternativa más factible (al menos a corto plazo) es mantener la infraestructura de VMware restante, por los siguientes motivos:

- **Estabilidad del entorno actual:** Con la implantación de la parte de cómputo en el nuevo clúster, el resto de sistemas no presenta un riesgo inminente a quedar comprometidos o inutilizados. Asimismo, el personal de administración posee un amplio nivel de madurez acerca de las comodidades y servicios ofrecidos, por lo que modificar el entorno obligaría a sufrir un proceso de aprendizaje de varias semanas hasta ser capaces de sacarle el máximo partido, o bien podría conllevar una pérdida parcial de funcionalidades.

- **Costes de re-implementación:** Al margen de tener que adquirir licencias adicionales para la integración de los hipervisores, también sería necesario transferir el contenido de los discos virtuales y copias de seguridad a la nueva infraestructura, además de adaptar el esquema de almacenamiento y red existente en dichos servidores, lo cual en un entorno de este tamaño supone a priori bastante tiempo añadido.
- **Garantía de rendimiento:** Tal y como hemos mencionado, no es posible afirmar rotundamente que el cambio de plataforma conlleve una mejora total de la capacidad técnica, ya que cada hipervisor está diseñado para funcionar de forma óptima en un segmento concreto del mercado.

En cualquier caso, resulta conveniente documentar algún protocolo a seguir si llegado el momento se decidiese aplicar una renovación completa de los sistemas o si los costes de mantenimiento de las licencias se volviesen insostenibles. En el anexo A.5 se incluye una propuesta al respecto.

7.2 Líneas de trabajo futuro

Tomando este proyecto como base, se plantean a continuación diversos enfoques de cara a ampliar su desarrollo:

- Escalar el clúster de SLURM para ver en detalle la interacción con múltiples nodos, así como implementar mecanismos de redundancia para los elementos de control y almacenamiento.
- Evaluar la capacidad de los hipervisores en términos de tolerancia a fallos y la eficacia de los procesos de recuperación.
- Incorporar en la comparativa alguna tecnología de despliegues basada en contenedores, con el objetivo de evaluar sus aptitudes frente a la virtualización por hipervisor.
- Considerar el uso de otros tipos de benchmark, como los de aplicaciones reales, para disponer de un conocimiento más extenso sobre el rendimiento de cada solución.
- Realizar pruebas de traslado de máquinas virtuales entre diferentes plataformas (e.g. VMware con Hyper-V), con el objetivo de determinar su eficiencia en un procedimiento de migración.

Apéndices

Apéndice A

Material adicional

A.1 Configuración de Slurm

```
1
2 # CLUSTER Y CONTROLADOR
3 ClusterName=citic-slurmtest
4 ControlMachine=maestroslurmtest
5 ControlAddr=<IP_maestro>
6 SlurmUser=slurm
7 SlurmctldPort=6817
8 SlurmdPort=6818
9 StateSaveLocation=/var/spool/slurm/ctl1d
10 SlurmdSpoolDir=/var/spool/slurm/d
11 SwitchType=switch/none
12 MpiDefault=none
13 SlurmctldPidFile=/var/run/slurmctld.pid
14 SlurmdPidFile=/var/run/slurmd.pid
15
16 # RMS
17 TaskPlugin=task/cgroup
18 ProctrackType=proctrack/cgroup
19 PluginDir=/usr/lib/slurm
20 ReturnToService=1
21 SlurmctldTimeout=300
22 SlurmdTimeout=300
23 InactiveLimit=0
24 MinJobAge=300
25 KillWait=30
26 Waittime=0
27
28 # POLÍTICAS DE ENCOLADO
29 SchedulerType=sched/backfill
30 SelectType=select/cons_tres
```

```

31 SelectTypeParameters=CR_Core_Memory
32 SlurmctldDebug=3
33 SlurmctldLogFile=/var/log/slurmctld.log
34 SlurmdDebug=3
35 SlurmdLogFile=/var/log/slurmd.log
36 JobCompType=jobcomp/none
37 JobAcctGatherType=jobacct_gather/cgroup
38 AccountingStorageEnforce=limits,associations
39 AccountingStorageTRES=gres/gpu,cpu,mem
40 DebugFlags=CPU_Bind,gres
41 #
42 # ACCOUNTING
43 AccountingStorageType=accounting_storage/slurmdbd
44 AccountingStorageHost=bdacc
45 AccountingStorageUser=slurm
46
47 # COMPATIBILIDAD GPU
48 GresTypes=gpu
49
50 # DEFINICION DE PARTICIONES
51 PartitionName=compute Nodes=comp1slurmtest Default=YES
      MaxTime=INFINITE State=UP
52 PartitionName=storage Nodes=almacenlurmtest Default=NO
      MaxTime=INFINITE State=DOWN
53 PartitionName=login Nodes=loginslurmtest Default=NO
      MaxTime=INFINITE State=DOWN
54 PartitionName=accounting Nodes=bdaccslurmtest Default=NO
      MaxTime=INFINITE State=DOWN
55
56 # DEFINICION DE NODOS
57 NodeName=maestroslurmtest NodeAddr=<IP_maestro> CPUs=2
      CoresPerSocket=1 ThreadsPerCore=1 RealMemory=3341 State=UNKNOWN
58 NodeName=almacenlurmtest NodeAddr=<IP_almacen> CPUs=2
      CoresPerSocket=1 ThreadsPerCore=1 RealMemory=3330 State=DOWN
59 NodeName=bdaccslurmtest NodeAddr=<IP_accounting> CPUs=2
      CoresPerSocket=1 ThreadsPerCore=1 RealMemory=3310 State=DOWN
60 NodeName=loginslurmtest NodeAddr=<IP_login> CPUs=2 CoresPerSocket=1
      ThreadsPerCore=1 RealMemory=3345 State=DOWN
61 NodeName=comp1slurmtest NodeAddr=<IP_computo> Gres=gpu:2 CPUs=24
      CoresPerSocket=6 ThreadsPerCore=2 RealMemory=515877 State=UNKNOWN

```

Listing A.1: Fichero slurm.conf

```

1
2 # AUTENTICACIÓN
3 AuthType=auth/munge

```

```
4  
5 # PARÁMETROS DEL DAEMON  
6 DbdAddr=localhost  
7 DbdHost=localhost  
8 SlurmUser=slurm  
9 DebugLevel=4  
10  
11 # PARÁMETROS DE LOGGING  
12LogFile=/var/log/slurm/slurmdbd.log  
13PidFile=/var/run/slurmdbd.pid  
14PluginDir=/usr/lib/slurm  
15  
16 # CONEXIÓN A BASE DE DATOS  
17  
18 # Si la BD está en otro host, incluir IP en StorageHost  
19StorageType=accounting_storage/mysql  
20StorageHost=localhost  
21StoragePass=slurmdbpass  
22StorageUser=slurm  
23StorageLoc=slurm_acct_db
```

Listing A.2: Fichero slurmdbd.conf

```
1 [Unit]  
2 Description=Slurm controller daemon  
3 After=network.target munge.service  
4 ConditionPathExists=/etc/slurm/slurm.conf  
5  
6 [Service]  
7 Type=forking  
8 EnvironmentFile=-/etc/sysconfig/slurmctl  
9 ExecStart=/usr/sbin/slurmctl $SLURMCTL_OPTIONS  
10 ExecReload=/bin/kill -HUP $MAINPID  
11 PIDFile=/var/run/slurmctl.pid  
12  
13 [Install]  
14 WantedBy=multi-user.target
```

Listing A.3: Fichero slurmctl.service

```
1 [Unit]  
2 Description=Slurm node daemon  
3 After=network.target munge.service  
4 ConditionPathExists=/etc/slurm/slurm.conf  
5  
6 [Service]  
7 Type=forking
```

```

8 EnvironmentFile=-/etc/sysconfig/slurmd
9 ExecStart=/usr/sbin/slurmd -d /usr/sbin/slurmstepd
$SLURMD_OPTIONS
10 ExecReload=/bin/kill -HUP $MAINPID
11 PIDFile=/var/run/slurmd.pid
12 KillMode=process
13 LimitNOFILE=51200
14 LimitMEMLOCK=infinity
15 LimitSTACK=infinity
16
17 [Install]
18 WantedBy=multi-user.target

```

Listing A.4: Fichero slurmd.service

```

1 [Unit]
2 Description=Slurm DBD accounting daemon
3 After=network.target munge.service
4 ConditionPathExists=/etc/slurm/slurmdbd.conf
5
6 [Service]
7 Type=forking
8 EnvironmentFile=-/etc/sysconfig/slurmdbd
9 ExecStart=/usr/sbin/slurmdbd $SLURMDBD_OPTIONS
10 ExecReload=/bin/kill -HUP $MAINPID
11 PIDFile=/var/run/slurmdbd.pid
12
13 [Install]
14 WantedBy=multi-user.target

```

Listing A.5: Fichero slurmdbd.service

```

1
2 # cgroup.conf
3 CgroupPlugin=cgroup/v1
4 ConstrainCores=yes
5 ConstrainDevices=yes
6 ConstrainRAMSpace=yes
7
8 # cgroup_allowed_devices_file.conf
9 /dev/null
10 /dev/urandom
11 /dev/zero
12 /dev/sda*
13 /dev/cpu/*/*
14 /dev/pts/*
15 /dev/nvidia*

```

Listing A.6: Ficheros cgroup.conf y cgroup_allowed_devices_file.conf

A.2 Despliegue de Prometheus/Grafana

```
1 global:
2     scrape_interval: 15s
3
4 scrape_configs:
5     - job_name: 'node_exporter'
6         static_configs:
7             - targets: ['comp1:9100', 'almacen:9100', 'bdacc:9100',
8                 'login:9100']
9
10    - job_name: 'slurm_exporter'
11        static_configs:
12            - targets: ['maestro:8080']
13
14    - job_name: 'dcgm_exporter'
15        static_configs:
16            - targets: ['comp1:9400']
```

Listing A.7: Fichero prometheus.yml

```
1 [Unit]
2 Description=Prometheus
3 After=network.target
4
5 [Service]
6 User=prometheus
7 ExecStart=/usr/local/bin/prometheus
8     --config.file=/etc/prometheus/prometheus.yml
9     --storage.tsdb.path=/var/lib/prometheus
10
11 [Install]
12 WantedBy=multi-user.target
```

Listing A.8: Fichero prometheus.service

A.3 Benchmarks y cargas de trabajo

```

1 #!/bin/bash
2
3 THREADS=$(nproc)      # En la versión de slurm, se usaría
4                   # directamente la variable SLURM_CPUS_PER_TASK
5
6 LOGFILE="/datasharing/jobs/benchmarks/results/benchmark_results.log"
7 echo "INITIALIZING BENCHMARK PROCESS AT $(date)" | tee -a $LOGFILE
8
9 # CPU Benchmark (Sysbench)
10 echo "BEGIN CPU BENCHMARK..." | tee -a $LOGFILE
11 sysbench cpu --threads=$THREADS run | tee -a $LOGFILE
12
13 # Memoria Benchmark (Sysbench)
14 echo "BEGIN MEMORY BENCHMARK..." | tee -a $LOGFILE
15 sysbench memory --threads=$THREADS run | tee -a $LOGFILE
16
17 # Almacenamiento Benchmark (fio)
18 echo "BEGIN STORAGE BENCHMARK..." | tee -a $LOGFILE
19 fio --name=test --rw=randrw --bs=4k --size=1G --numjobs=4
20     --time_based=1 --runtime=60 | tee -a $LOGFILE
21
22 # Red Benchmark (iperf3)
23 echo "BEGIN NETWORK BENCHMARK..." | tee -a $LOGFILE
24 iperf3 -c 10.56.67.191 -t 10 | tee -a $LOGFILE
25
26 echo "COMPLETED BENCHMARK PROCESS AT $(date)" | tee -a $LOGFILE

```

Listing A.9: Benchmark de rendimiento (compbench.sh)

```

1 import tensorflow as tf
2 from keras import layers, models
3 import time
4
5 # CARGAR CONJUNTO DE DATOS MNIST
6 (train_images, train_labels), (test_images, test_labels) =
7     tf.keras.datasets.mnist.load_data()
8
9 # PREPOCESAR DATOS
10 train_images = train_images.reshape((train_images.shape[0], 28, 28,
11     1)).astype('float32') / 255
12 test_images = test_images.reshape((test_images.shape[0], 28, 28,
13     1)).astype('float32') / 255
14
15 # DEFINIR MODELO
16 model = models.Sequential([
17     layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28,
18         28, 1))
19 ])
20
21 # COMPILEAR MODELO
22 model.compile(optimizer='adam',
23                 loss='sparse_categorical_crossentropy',
24                 metrics=['accuracy'])
25
26 # ENTRENAR MODELO
27 history = model.fit(train_images, train_labels, epochs=10,
28                     validation_data=(test_images, test_labels))
29
30 # EVALUAR MODELO
31 print("Test accuracy: ", history.history['val_accuracy'][-1])

```

```
28, 1)),  
15 layers.MaxPooling2D((2, 2)),  
16 layers.Conv2D(64, (3, 3), activation='relu'),  
17 layers.MaxPooling2D((2, 2)),  
18 layers.Conv2D(64, (3, 3), activation='relu'),  
19 layers.Flatten(),  
20 layers.Dense(64, activation='relu'),  
21 layers.Dense(10, activation='softmax')  
22 ])  
23  
24 # COMPIALAR MODELO  
25 model.compile(optimizer='adam',  
26                 loss='sparse_categorical_crossentropy',  
27                 metrics=['accuracy'])  
28  
29 # EJECUTAR ENTRENAMIENTO  
30 start_time = time.time()  
31  
32 model.fit(train_images, train_labels, epochs=5, batch_size=64,  
            validation_data=(test_images, test_labels))  
33  
34 end_time = time.time()  
35  
36 print(f"Tiempo de entrenamiento: {end_time - start_time} segundos")
```

Listing A.10: Modelo de prueba de Tensorflow (tf-train.py)

A.4 Extracción de resultados del benchmark de rendimiento

```
1 import re  
2 import csv  
3 import os  
4 import argparse  
5  
6 # CONFIGURAR ARGUMENTOS SCRIPT  
7 parser = argparse.ArgumentParser(description='Extraer métricas de  
        benchmark desde archivos de texto.')  
8 parser.add_argument('input_folder', help='Ruta a la carpeta con  
                     archivos .txt de benchmark')  
9 parser.add_argument('--output', default='benchmark_comparison.csv',  
                    help='Nombre del archivo de salida CSV')  
10 args = parser.parse_args()  
11  
12 # FUNCIÓN DE EXTRACCIÓN  
13 def extract_data_from_file(filepath):
```

```

14 data = {
15     'filename': os.path.basename(filepath),
16     'CPU events/s': None,
17     'CPU latency avg (ms)': None,
18     'Memory throughput (MiB/s)': None,
19     'Disk IOPS (avg)': None,
20     'Disk Read Latency (avg, us)': None,
21     'Disk Write Latency (avg, us)': None,
22     'Network Bandwidth (Mbps)': None,
23     'Network Retransmits': None
24 }
25
26 with open(filepath, 'r') as f:
27     content = f.read()
28
29 # CPU
30 cpu_events = re.search(r'CPU speed:\s+events per second:\s+([\d.]+)', content)
31 if cpu_events: data['CPU events/s'] = float(cpu_events.group(1))
32 cpu_latency = None
33 lines = content.splitlines()
34 for i, line in enumerate(lines):
35     if "Latency (ms):" in line:
36         for j in range(i+1, i+6): # Revisa las siguientes pocas
37             if "avg:" in lines[j]:
38                 match = re.search(r'avg:\s+([\d.]+)', lines[j])
39                 if match:
40                     cpu_latency = float(match.group(1))
41                     break
42             break
43 if cpu_latency is not None:
44     data['CPU latency avg (ms)'] = cpu_latency
45
46 # Memoria
47 mem_throughput = re.search(r'([\d.]+) MiB/sec', content)
48 if mem_throughput: data['Memory throughput (MiB/s)'] =
49     float(mem_throughput.group(1))
50
51 # Disco
52 disk_iops = re.findall(r'IOPS=(\d+), BW=.+\n.*?avg=([\d.]+)', content)
53 if disk_iops:
54     iops_vals = [int(i[0]) for i in disk_iops]

```

```
55     read_latencies = [float(i[1]) for i in
56         disk_iops[:len(disk_iops)//2]]
57     write_latencies = [float(i[1]) for i in
58         disk_iops[len(disk_iops)//2:]]
59     data['Disk IOPS (avg)'] = sum(iops_vals) / len(iops_vals)
60     data['Disk Read Latency (avg, us)'] = sum(read_latencies) /
61         len(read_latencies)
62     data['Disk Write Latency (avg, us)'] = sum(write_latencies) /
63         len(write_latencies)
64
65     # Red
66     network_match = re.search(
67         r'[\s*5\].*?0\.00-10\.00\s+sec\s+[\d.]+\s+(?:G|M)Bytes\s+
68         ([\d.]+)\s+(G|M)bits/sec\s+(\d+)',
69         content, re.DOTALL)
70
71     if network_match:
72         value = float(network_match.group(1))
73         unit = network_match.group(2)
74         retrans = int(network_match.group(3))
75
76         if unit == 'G':
77             bandwidth_mbps = value * 1000
78         else:
79             bandwidth_mbps = value
80
81     data['Network Bandwidth (Mbps)'] = bandwidth_mbps
82     data['Network Retransmits'] = retrans
83
84     return data
85
86
87 # RECOPILAR ARCHIVOS
88 benchmark_files = [os.path.join(args.input_folder, f)
89                     for f in os.listdir(args.input_folder) if
90                     f.endswith('.txt')]
91
92 if not benchmark_files:
93     print("No se encontraron archivos .txt en la carpeta especificada.")
94     exit(1)
95
96 # PROCESAR Y ALMACENAR
97 all_data = [extract_data_from_file(file) for file in benchmark_files]
98 with open(args.output, 'w', newline='') as csvfile:
99     fieldnames = list(all_data[0].keys())
100    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
```

```

96 writer.writeheader()
97 writer.writerows(all_data)
98
99 print(f"\n Resultados guardados en '{args.output}'")

```

Listing A.11: Código benchtocs.py para recopilar métricas de compbench

A.5 Propuesta de plan de migración a largo plazo

Esta parte pretende reproducir la documentación de un protocolo estructurado para el traslado de la infraestructura de virtualización de un centro de datos, adaptado a las características de las instalaciones del CITIC.

A.5.1 Alcance

Participan de este procedimiento aquellos recursos que constituyen la operativa fundamental de la organización (servidores, máquinas virtuales, elementos de almacenamiento y red), a excepción del entorno destinado a las ejecuciones de cómputo. El objetivo es lograr eventualmente una reubicación completa de la arquitectura de VMware a otra plataforma con un menor coste asociado, pasando previamente por un período de coexistencia entre ambos ecosistemas.

Por otro lado, se considera la siguiente división de etapas: **Inventariado, planificación del flujo, traslado de ensayo y fase de validación**.

A.5.2 Composición inicial

A fecha de inicio del proyecto, la infraestructura engloba 13 servidores repartidos entre 4 agrupaciones o clúster con **más de 300 máquinas virtuales**, donde cerca del 75% están destinadas a labores de investigación y algo menos de un 10% se utilizan como rol administrativo de varios servicios estratégicos del centro. El resto lo conforman despliegues de aplicaciones web u otras tareas con fines demostrativos. Pese a ello, solamente en torno a un 80% están operativas, por lo que se reduce ligeramente la lista de servicios a tener en cuenta.

En lo que respecta al almacenamiento, se incluye una **red SAN** de al menos 4 cabinas de discos en combinación con un servidor para copias de seguridad remotas mediante **NFS**. Por otra parte, se dispone de alrededor de **50 segmentos de red o VLANs** (contando la anterior) de los cuales 3 son para gestión y 1 para DMZ (Zona Desmilitarizada).

Sobre la distribución de los componentes, no es necesario realizar ningún cambio ya que el traslado tiene lugar únicamente a nivel lógico. En caso contrario, sería preciso hacer un estudio en profundidad de la nueva localización y la posterior instalación de los componentes, sobre todo en lo referente al ciclo de enfriamiento de los equipos.

A.5.3 Planificación

En vista del esquema previo, se considera la siguiente secuencia de pasos para llevar a cabo una correcta integración:

1. Realización de copias de seguridad de las máquinas con una mayor criticidad (e.g. servicios web y de administración), así como aquellas involucradas en actividades de elevado impacto entre los grupos de investigación.
2. Selección de candidatos para la fase de prueba, preferiblemente que no sean una parte sustancial de la producción, atendiendo a factores como fecha de creación, acceso a los recursos, tiempo de inactividad y/o sensibilidad en su contenido. Ante todo, han de ser ejemplos heterogéneos y representativos de las principales cargas de trabajo de la infraestructura. También deben incluir una copia de seguridad si no se llegó a efectuar en el primer paso.
3. Replicación de las conexiones a las VLANs y almacenes de datos a configurar en la nueva infraestructura. Sería conveniente organizar estos últimos para optimizar dicho proceso, por ejemplo incluyendo los discos virtuales de los candidatos en un mismo almacén.
4. Determinación del instante a partir del cual se delega la generación de futuras máquinas en el entorno destino. Esto dependerá del histórico que posea la entidad sobre la frecuencia de aparición de esta clase de peticiones. Analizando las estadísticas del gestor de incidencias usado por el [CITIC](#), concluimos que esta operación ocurre en promedio cada 2-3 semanas.

A.5.4 Migración de prueba

Una vez establecidos los puntos clave, lo siguiente es tomar una muestra aleatoria de las máquinas escogidas y resolver cualquier inconsistencia que pudiesen tener en la infraestructura base, para acto seguido registrar su estado, asignación de recursos, dependencias e interacciones principales con otros servicios de carácter externo (servidores de ficheros, bases de datos, aplicaciones en segundo plano, etc.). Además, se deberá notificar con cierta antelación a los usuarios afectados sobre su parada por mantenimiento y qué implicaciones trae consigo.

Por último se procede a exportar las máquinas ya inactivas (y con todas sus instantáneas¹ eliminadas) en un formato reconocible por la nueva plataforma, dentro de la cual se creará una instancia de configuraciones equivalentes.

¹ Una instantánea o *snapshot* hace referencia a la captura del estado de una máquina en un punto concreto.

A.5.5 Comprobaciones post-migración

Al llevar a cabo el arranque de las máquinas virtuales en el nuevo entorno, se debe monitorizar su rendimiento a lo largo de varios días, así como verificar la integridad de los datos y que se conserva el acceso a todas sus funcionalidades. También es conveniente que los usuarios finales aporten comentarios de retroalimentación sobre su experiencia para validar la interoperabilidad del ecosistema y tratar de hallar nuevos errores.

Lista de acrónimos

BOE Boletín Oficial del Estado. [20](#)

CITIC Centro de Investigación en Tecnologías de Información y Comunicación. [1](#), [27](#), [31](#), [71](#), [73](#), [74](#), [88](#), [89](#)

CLI Command Line Interface. [49](#)

CPD Centro de Procesamiento de Datos. [1](#), [2](#), [8](#), [9](#), [27](#), [31](#), [74](#)

GPU Graphical Processing Unit. [1](#), [7](#), [25](#), [28](#), [31](#), [48](#), [59](#), [60](#), [70](#)

HPC Computación de Altas Prestaciones. [1](#), [5](#), [6](#), [29](#)

NFS Network File System. [35](#), [88](#)

PYME Pequeñas y Medianas Empresas. [27](#)

RMS Resource Management System. [11](#)

SAI Sistema de Alimentación Ininterrumpida. [9](#)

SLURM Simple Linux Utility for Resource Management. [24](#), [25](#), [32](#), [33](#), [36](#), [37](#), [44](#), [46](#), [49](#), [58](#), [63](#), [65](#), [66](#), [68–71](#), [75](#)

TI Tecnologías de la Información. [9](#), [31](#)

TIA Telecommunications Industry Association. [9](#)

Glosario

benchmark Un banco de pruebas representa al conjunto de modelos, parámetros y/o resultados que se toman como punto de partida de cara a la evaluación de elementos de su misma categoría. [6](#), [28](#), [75](#)

dashboard Representación gráfica de métricas relevantes en los objetivos de un proceso particular. [26](#)

diagrama de Gantt Consiste en una visualización esquematizada sobre la organización de las tareas de un proyecto y su relación en la escala de tiempo. [19](#), [20](#), [28](#), [V](#)

firewall También llamado cortafuegos, componente *software* o *hardware* utilizado como medida de seguridad para restringir las comunicaciones entre dispositivos a nivel interno y externo. [50](#)

framework Agrupación de herramientas que definen la estructura o marco principal de un sistema. [24](#), [27](#)

plug-in Complemento que extiende la funcionalidad base de un programa sin manipular el código original. [24](#), [37](#), [44](#)

standalone Capacidad de cualquier elemento computacional de operar por sí mismo. [27](#)

Bibliografía

- [1] V. L. C. von Zitzewitz, “Nvidia’s bet on artificial intelligence,” Master’s thesis, Universidade NOVA de Lisboa (Portugal), 2024.
- [2] K. Kant, “Data center evolution: A tutorial on state of the art, issues, and challenges,” *Computer Networks*, vol. 53, no. 17, pp. 2939–2965, 2009.
- [3] T. Sterling, M. Brodowicz, and M. Anderson, *High performance computing: modern systems and practices*. Morgan Kaufmann, 2017.
- [4] A. J. Hey, “High performance computing-past, present and future,” *Computing and Control Engineering*, vol. 8, no. 1, pp. 33–42, 1997.
- [5] W. Chan, A. D’Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, “A brief history of hpc simulation and future challenges.”
- [6] D. Barkai, *Unmatched: 50 Years of Supercomputing*. CRC Press, 2023.
- [7] D. Reed, D. Gannon, and J. Dongarra, “Reinventing high performance computing: challenges and opportunities,” *arXiv preprint arXiv:2203.02544*, 2022.
- [8] H. Hussain, S. U. R. Malik, A. Hameed, S. U. Khan, G. Bickler, N. Min-Allah, M. B. Qureshi, L. Zhang, W. Yongji, N. Ghani *et al.*, “A survey on resource allocation in high performance distributed computing systems,” *Parallel Computing*, vol. 39, no. 11, pp. 709–736, 2013.
- [9] N. Sadashiv and S. D. Kumar, “Cluster, grid and cloud computing: A detailed comparison,” in *2011 6th international conference on computer science & education (ICCSE)*. IEEE, 2011, pp. 477–482.
- [10] D. Milojicic, P. Faraboschi, N. Dube, and D. Roweth, “Future of hpc: Diversifying heterogeneity,” in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 276–281.

- [11] A. Benhari, D. Trystram, F. Dufossé, Y. Denneulin, and F. Despres, “Green hpc: An analysis of the domain based on top500,” *arXiv preprint arXiv:2403.17466*, 2024.
- [12] E. A. Huerta, A. Khan, E. Davis, C. Bushell, W. D. Gropp, D. S. Katz, V. Kindratenko, S. Koric, W. T. Kramer, B. McGinty *et al.*, “Convergence of artificial intelligence and high performance computing on nsf-supported cyberinfrastructure,” *Journal of Big Data*, vol. 7, pp. 1–12, 2020.
- [13] C. Silvano, D. Ielmini, F. Ferrandi, L. Fiorin, S. Curzel, L. Benini, F. Conti, A. Garofalo, C. Zambelli, E. Calore *et al.*, “A survey on deep learning hardware accelerators for heterogeneous hpc platforms,” *arXiv preprint arXiv:2306.15552*, 2023.
- [14] Y. Fan, “Job scheduling in high performance computing,” *arXiv preprint arXiv:2109.09269*, 2021.
- [15] D. A. Reed and J. Dongarra, “Exascale computing and big data,” *Communications of the ACM*, vol. 58, no. 7, pp. 56–68, 2015.
- [16] D. Rosendo, A. Costan, P. Valduriez, and G. Antoniu, “Distributed intelligence on the edge-to-cloud continuum: A systematic literature review,” *Journal of Parallel and Distributed Computing*, vol. 166, pp. 71–94, 2022.
- [17] K. Bilal, S. U. Khan, L. Zhang, H. Li, K. Hayat, S. A. Madani, N. Min-Allah, L. Wang, D. Chen, M. Iqbal *et al.*, “Quantitative comparisons of the state-of-the-art data center architectures,” *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1771–1783, 2013.
- [18] C. Harvey, “What is a data center? history, design, cooling & types,” 2017. [En línea]. Disponible en: <https://www.datamation.com/data-center/what-is-data-center/>
- [19] S. D. Lowe, J. Green, and D. Davis, “Building a modern data center,” *Atlantis, Published by Published by ActualTech Media In Partnership with Atlantis Computing, ActualTech Media Okatie Village Ste*, pp. 103–157, 2016.
- [20] A. Bartels, “Data center evolution: 1960 to 2000,” 2011. [En línea]. Disponible en: <https://web.archive.org/web/20181024232055/https://blog.rackspace.com/datacenter-evolution-1960-to-2000>
- [21] A. Varasteh and M. Goudarzi, “Server consolidation techniques in virtualized data centers: A survey,” *IEEE Systems Journal*, vol. 11, no. 2, pp. 772–783, 2015.
- [22] V. Matko, B. Brezovec, and M. Milanović, “Intelligent monitoring of data center physical infrastructure,” *Applied Sciences*, vol. 9, no. 23, p. 4998, 2019.

- [23] N. Jain and S. Choudhary, “Overview of virtualization in cloud computing,” in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*. IEEE, 2016, pp. 1–4.
- [24] A. Rashid and A. Chaturvedi, “Virtualization and its role in cloud computing environment,” *International Journal of Computer Sciences and Engineering*, vol. 7, no. 4, pp. 1131–1136, 2019.
- [25] D. Ageyev, O. Bondarenko, T. Radivilova, and W. Alfroukh, “Classification of existing virtualization methods used in telecommunication networks,” in *2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*. IEEE, 2018, pp. 83–86.
- [26] B. Golden, *Virtualization for dummies*. John Wiley & Sons, 2007.
- [27] S. Singh and I. Chana, “A survey on resource scheduling in cloud computing: Issues and challenges,” *Journal of grid computing*, vol. 14, pp. 217–264, 2016.
- [28] S. Prabhakaran, “Dynamic resource management and job scheduling for high performance computing,” *Technische Universität Darmstadt*, 2016.
- [29] A. Wadhonkar and D. Theng, “A survey on different scheduling algorithms in cloud computing,” in *2016 2nd International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB)*. IEEE, 2016, pp. 665–669.
- [30] A. Reuther, C. Byun, W. Arcand, D. Bestor, B. Bergeron, M. Hubbell, M. Jones, P. Michaleas, A. Prout, A. Rosa *et al.*, “Scalable system scheduling for hpc and big data,” *Journal of Parallel and Distributed Computing*, vol. 111, pp. 76–92, 2018.
- [31] HPCWiki, “Scheduling basics,” 2024. [En línea]. Disponible en: https://hpc-wiki.info/hpc/Scheduling_Basics
- [32] J. Hall, A. Lathi, D. Lowenthal, and T. Patki, “A comparison of backfilling and coscheduling for high-performance computing,” Lawrence Livermore National Laboratory (LLNL), Livermore, CA (United States), Tech. Rep., 2023.
- [33] V. Gisbert Soler and O. Raissouni, “Benchmarking, herramienta de control de calidad y mejora continua,” *3C Empresa*, vol. 3, no. 20, pp. 217–233, 2014.
- [34] F. Del Giorgio Solfa, “Benchmarking en el sector público,” *Industry Consulting*, 2012.
- [35] J. v. Kistowski, J. A. Arnold, K. Huppler, K.-D. Lange, J. L. Henning, and P. Cao, “How to build a benchmark,” in *Proceedings of the 6th ACM/SPEC international conference on performance engineering*, 2015, pp. 333–336.

- [36] A. Srivastava, S. Bhardwaj, and S. Saraswat, “Scrum model for agile methodology,” in *2017 International Conference on Computing, Communication and Automation (ICCCA)*. IEEE, 2017, pp. 864–869.
- [37] S. Sachdeva, “Scrum methodology,” *Int. J. Eng. Comput. Sci*, vol. 5, no. 16792, pp. 16 792–16 800, 2016.
- [38] “Boe: Resolución de tablas salariales en 2024 para xx convenio colectivo nacional de empresas de ingeniería, oficinas de estudios técnicos, inspección, supervisión y control técnico y de calidad,” 2024. [En línea]. Disponible en: <https://www.boe.es/boe/dias/2024/03/23/pdfs/BOE-A-2024-5873.pdf>
- [39] A. B. Yoo, M. A. Jette, and M. Grondona, “Slurm: Simple linux utility for resource management,” in *Workshop on job scheduling strategies for parallel processing*. Springer, 2003, pp. 44–60.
- [40] Schedmd, “Slurm workload manager documentation,” 2024. [En línea]. Disponible en: <https://slurm.schedmd.com/documentation.html>
- [41] D. Bartholomew, “Mariadb vs. mysql,” *Dostopano*, vol. 7, no. 10, p. 2014, 2012.
- [42] prometheus, “Exporter for machine metrics,” 2025. [En línea]. Disponible en: https://github.com/prometheus/node_exporter
- [43] vpenso, “Prometheus exporter for performance metrics from slurm,” 2025. [En línea]. Disponible en: <https://github.com/vpenso/prometheus-slurm-exporter>
- [44] NVIDIA, “Nvidia gpu metrics exporter for prometheus leveraging dcgm,” 2025. [En línea]. Disponible en: <https://github.com/NVIDIA/dcgm-exporter>
- [45] H. Fayyad-Kazan, L. Perneel, and M. Timmerman, “Benchmarking the performance of microsoft hyper-v server, vmware esxi and xen hypervisors,” *Journal of Emerging Trends in Computing and Information Sciences*, vol. 4, no. 12, pp. 922–933, 2013.
- [46] B. Asvija, R. Eswari, and M. Bijoy, “Security in hardware assisted virtualization for cloud computing—state of the art issues and challenges,” *Computer Networks*, vol. 151, pp. 68–92, 2019.
- [47] Proxmox, “Proxmox ve overview,” 2025. [En línea]. Disponible en: <https://www.proxmox.com/en/products/proxmox-virtual-environment/overview>
- [48] C. Dunlap, “Munge uid ‘n’ gid emporium,” 2024. [En línea]. Disponible en: <https://github.com/dun/munge/wiki/Installation-Guide>

- [49] D. Pousa and J. Rufino, “Benchmarking of bare metal virtualization platforms on commodity hardware,” in *14th International Conference on Applied Computing (AC 2017)*. IADIS, 2017, pp. 205–212.
- [50] J. Baumgartner, C. Lillo, and S. Rumley, “Performance losses with virtualization: Comparing bare metal to vms and containers,” in *International Conference on High Performance Computing*. Springer, 2023, pp. 107–120.
- [51] Microsoft, “Windows server pricing,” 2024. [En línea]. Disponible en: <https://www.microsoft.com/en-us/windows-server/pricing>
- [52] Proxmox, “Pricing for subscriptions plans,” 2025. [En línea]. Disponible en: <https://www.proxmox.com/en/products/proxmox-virtual-environment/pricing>
- [53] Broadcom, “Licensing and subscription in vsphere,” 2025. [En línea]. Disponible en: <https://techdocs.broadcom.com/us/en/vmware-cis/vsphere/vsphere/8-0/vcenter-and-host-management-8-0/license-management-host-management/licensing-for-products-in-vsphere-host-management.html>
- [54] RedRessCompliance, “Broadcom vmware licensing and subscription changes explained,” 2025. [En línea]. Disponible en: <https://redresscompliance.com/broadcom-vmware-licensing-and-subscription-changes-explained/>
- [55] vINCEPTION, “Vmware: Licensing changes for vmware products,” 2024. [En línea]. Disponible en: <https://vinception.fr/vmware-licensing-changes-for-vmware-products/>