Universidad Técnica Estatal de Quevedo

Estudiante: Cedeño Orlando, Vilcacundo Jordy, Robalino Bryan y Orrala William

Curso: Ingeniería en Software 7mo “A”.

Tema: Computación Paralela y Distribuida.

Asignatura: Aplicaciones Distribuidas.

Docente: Guerrero Ulloa Gleiston Cicerón.

Índice

Tabla de Ilustraciones

1. Introducción

La computación paralela y la computación distribuida son áreas cruciales en ciencias de la computación, enfocadas en el procesamiento simultáneo de tareas para mejorar el rendimiento y la eficiencia [1]. El desarrollo constante del hardware de computadoras hacia una mayor capacidad paralela impulsa la demanda de desarrolladores de software con experiencia en la escritura de programas paralelos[1],[2]. Esta tendencia subraya la importancia de integrar la enseñanza del paralelismo en los cursos básicos de informática, lo cual es esencial para equipar a los futuros ingenieros de software con las habilidades necesarias para enfrentar desafíos contemporáneos en el campo [2].

En el ámbito académico, se ha prestado atención a la paralelización de algoritmos complejos, como la programación estocástica dual dinámica (SDDP), una técnica utilizada en la toma de decisiones bajo incertidumbre [3]. Los esfuerzos de investigación se han centrado en desarrollar esquemas de paralelización que permitan una ejecución más eficiente, ya sea paralelizando el proceso de cálculo por escenario o por nodo [4].

Más allá del ámbito académico, la computación paralela también ha encontrado aplicaciones prácticas significativas, como en el campo de la radioterapia. Aquí, los marcos de computación paralela se utilizan para optimizar la planificación del tratamiento de pacientes, considerando simultáneamente múltiples factores como la dosis, la tasa de dosis y la transferencia lineal de energía (LET) [5].

Finalmente, en el contexto de los sistemas distribuidos, se han explorado técnicas como la computación paralela por clústeres. Estas técnicas buscan mejorar el rendimiento de los sistemas distribuidos mediante la coordinación y el procesamiento paralelo de tareas en múltiples nodos de un clúster, aprovechando así la potencia de procesamiento colectiva de varios ordenadores conectados en red [6].

2. Computación paralela

2.1. Arquitecturas según Instrucciones y Datos

Esto generalmente se refiere a la arquitectura de un sistema informático que se basa en la separación de las instrucciones y los datos en el nivel de diseño.

Las arquitecturas según instrucciones ISA Con el creciente valor de opercaciones en datos dispersos, SparseCore se convirtió en la primera extensión de procesador de

propósito general. Su flexibilidad permite la aceleración de modelos de código complejos y algoritmos variables, superando las limitaciones de los aceleradores actuales diseñados para aplicaciones especiales. Este ejemplo extiende la estructura general (ISA) al elemento básico utilizando un ISA innovador específico del tráfico. [7]

En el contexto de la arquitectura según datos, se destaca la importancia de la informática en la era del big data y las tecnologías avanzadas de aprendizaje automático. Enfatiza la necesidad de diseñar sistemas de TI para satisfacer eficazmente estas necesidades y enfatiza la innovación en infraestructura heterogénea y modelos de computación en la nube. Además, señala la importancia de diseñar sistemas que cubran todo el flujo de trabajo, incluida la gestión de datos, y destaca la aceleración del hardware como clave para mejorar el rendimiento cognitivo. [8]

Un ejemplo muestra que los aceleradores espaciales basados en "instrucciones de dispositivo" pueden alcanzar un rendimiento estandarizado por unidad de área 8 veces mayor que el de los procesadores tradicionales de uso general. En comparación con el enfoque espacial basado en contadores de programas, el control basado en "instrucción de activación" reduce significativamente la cantidad de instrucciones estáticas y dinámicas en la ruta crítica, lo que resulta en una aceleración de 2,0 veces. [9]

2.1.1. Arquitectura paralela SIMD

La arquitectura paralela de SIMD tiene como objetivo optimizar el rendimiento de los operadores morfológicos difusos mediante el uso de la capacidad de realizar múltiples operaciones en conjuntos de datos al mismo tiempo, mejorando así la eficiencia del procesamiento de imágenes [10]. Este tema también aborda la eficiente ejecución de computaciones en streaming en arquitecturas SIMD, que a menudo presentan un importante paralelismo de datos, los problemas intermitentes con estas aplicaciones son comunes. La implementación del sistema MERCATOR en las GPU NVIDIA muestra mejoras empíricas de rendimiento para aplicaciones de flujo irregular [11].

Se busca, ademas, mejorar el rendimiento de microprocesadores heterogéneos mediante paralelización MIMD-SIMD anidada. Bucles de datos paralelos regulares se anidan dentro de un bucle externo paralelo con código irregular. Esta estrategia logra un

notable aumento de velocidad en comparación con la ejecución secuencial, superando en rendimiento a enfoques que utilizan solo la CPU o la GPU [12].

2.1.2. Arquitectura paralela MIMD

La arquitectura paralela MIMD utiliza múltiples procesadores independientes para ejecutar instrucciones y trabajar en diferentes conjuntos de datos simultáneamente, ejemplo.El potencial de los sistemas de memoria híbrida (DRAM y NVRAM) para almacenar y procesar eficientemente datos columnares en la memoria principal, se demuestra que la combinación de implementaciones SIMD en ejecuciones multi-hilo puede aumentar significativamente el rendimiento de consultas concurrentes. Se propone un enfoque adaptativo de cóctel SIMD-MIMD para optimizar esta interacción, con un costo temporal insignificante [13].

Se presenta una arquitectura que facilita el uso de múltiples paneles Parallella en un solo sistema con la capacidad de agregar dispositivos informáticos adicionales. Se centra en aplicación de sistemas multiagente y presenta escenarios seleccionados que se pueden implementar fácilmente utilizando las capacidades de múltiples unidades MIMD [12].

Un ejemplo es del lenguaje de modelado paralelo para sistemas dinámicos complejos, destacando las limitaciones de las herramientas de simulación paralela en comparación con los lenguajes secuenciales, ademas, la conexión entre el análisis de sistemas secuenciales y el paradigma MIMD paralelo, que describe cómo las especificaciones BO se transforman en procesos MIMD, donde cada bloque realiza una operación correspondiente [14].

2.1.3. Procesamiento en paralelo a nivel de instrucciones y datos

Una herramienta es CASPER intenta superar la resistencia a la introducción de nuevos marcos reescribiendo automáticamente programas secuenciales en las DSL o API de esos marcos, A través de una interfaz de navegador, los usuarios ingresan el código fuente de Java y CASPER redirige automáticamente para ejecutar Apache Spark, mostró las

capacidades de optimización de CASPER en programas de visualización de datos y procesamiento de imágenes, ejecutando simultáneamente las versiones optimizadas y secuenciales en la nube para que los espectadores las comparen en tiempo real [15].

La utilidad de las GPU como coprocesadores de bases de datos para resolver problemas de deriva del flujo de control causados por una distribución desigual de los datos se han propuesto técnicas para equilibrar los efectos de divergencia y aumentar la eficiencia del procesamiento, el compilador de consultas DogQC supera a otros compiladores y logra mejoras significativas en tiempos de ejecución más cortos en comparación con los compiladores de GPU y los sistemas de CPU[2].

Este es un ejemplo que resuelve el problema del "muro de memoria" mediante el uso del procesamiento en memoria (PIM), que procesa datos directamente en la memoria sin transferirlos entre la CPU y la memoria, se centra en la optimización de redes neuronales binarias (BNN) utilizando PIM AND, NOT y operaciones de números de población utilizando instrucciones RISC-V personalizadas, un nuevo diseño plegable de BNN y un diseño de memoria mejorado, Se mostraron mejoras significativas, siendo la inferencia de un extremo a otro del modelo BNN 57,3 veces más rápida que los sistemas basados en CPU [16].

2.2. Arquitectura según Distribuciones de memoria

2.2.1. Arquitecturas de memoria compartida

Las arquitecturas de memoria compartida son un modelo de programación utilizado en sistemas multinúcleo como x86, Power y ARM. Estas arquitecturas permiten múltiples tareas para asignar y acceder a los datos a pesar de su localización remota, lo que las hace convenientes para sistemas heterogéneos distribuidos. También permiten la implementación de aplicaciones en arquitecturas distribuidas y facilitan la programabilidad de hardware complejo [17]. En el contexto del aprendizaje profundo, las arquitecturas de memoria compartida se han utilizado para entrenar redes neuronales profundas en CPUs multinúcleo y dispositivos GPU, comparando la eficiencia paralela de implementaciones asíncronas y sincrónicas [18]. Además, las arquitecturas de memoria compartida se han aprovechado para diseñar implementaciones de algoritmos de alto rendimiento para multiplicar matrices de bits en sistemas de múltiples

aceleradores contemporáneos, optimizando el tiempo, la eficiencia energética y la escalabilidad [19].

La memoria compartida es un cómodo modelo de programación en el que un conjunto de tareas (procesos, hilos) pueden acceder (asignar, leer, escribir) a un espacio de memoria común. Esto es bastante sencillo en la arquitectura clásica Von-Neumann, en la que las memorias físicas se comparten entre las unidades de procesamiento. En la que las memorias físicas se comparten entre las unidades de procesamiento. Sin embargo, cuando arquitecturas distribuidas, las unidades de procesamiento no pueden acceder directamente memorias remotas utilizando un espacio de direcciones local. Se necesitan sistemas intermedios de hardware o para gestionar de forma transparente las solicitudes de acceso [18].

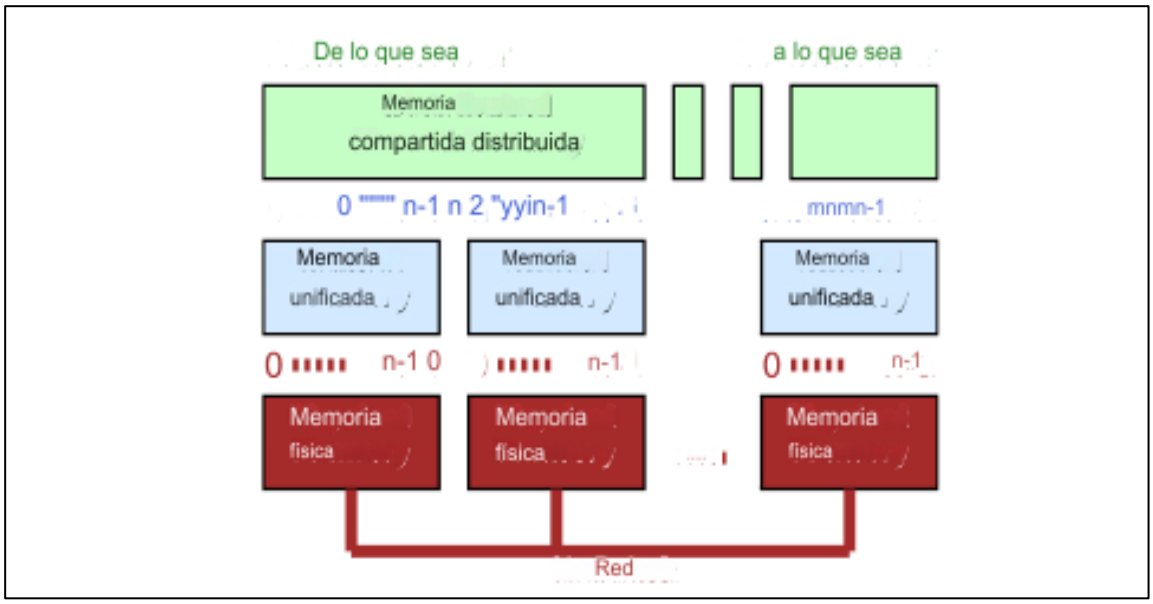

Ilustración 1: [18] Atomique, "Memoria Compartida Distribuida", 2009

El modelo de programación define la semántica de un programa, da garantías sobre el comportamiento del programa pero no especifica cómo se ejecuta. Definir la ejecución de un programa es el papel del modelo de ejecución. Debe ajustarse a la semántica de alto nivel descrita por el modelo de programación, pero es libre de elegir la implementación subyacente. Desacoplar los modelos de programación y ejecución es importante cuando se habla de rendimiento, porque aunque el modelo de programación impone restricciones a la ejecución la ejecución, diferentes implementaciones pueden dar lugar a perfiles de perfiles de rendimiento muy diferentes [20].

2.2.2. Arquitecturas de memoria distribuida

Las arquitecturas de memoria distribuida son un tipo de arquitectura de computadora que permite la asignación y el acceso de datos a través de múltiples nodos en un sistema distribuido. Estas arquitecturas proporcionan beneficios como el escalado independiente de los recursos de cómputo y memoria, así como un modelo de falla independiente en el que las fallas de cómputo no afectan a los datos residentes en la memoria distribuida [21]. Son particularmente útiles en arquitecturas heterogéneas distribuidas, donde se utilizan diferentes tipos de hardware, como CPU y GPU [22][23]. Se han propuesto diversas técnicas y optimizaciones para abordar los desafíos asociados con las arquitecturas de memoria distribuida, como la desagregación de memoria, implementaciones eficientes fuera de memoria y el uso de modelos de programación basados en tareas y sistemas de tiempo de ejecución para la tolerancia a fallas [18]. Estos enfoques tienen como objetivo mejorar la utilización de la memoria, reducir la latencia y mejorar la escalabilidad y el performance de los sistemas de memoria distribuida[24].

La desagregación de recursos es un enfoque popular para especializar los recursos asignados a una carga de trabajo recursos asignados a una carga de trabajo. Cada vez se presta más atención a aprovechar los recursos de memoria fuera de los nodos de nodos de computación tradicionales. Aquí nos centramos en la memoria en soluciones de memoria remota en las que los nodos de computación comparten su memoria con otros nodos de computación. En las organizaciones de sistemas de memoria desagregada, la memoria que es de los nodos de cálculo se expone a través de una red rápida [23].

2.2.3. Sistemas NUMA

Los sistemas NUMA son arquitecturas que tienen acceso no uniforme a la memoria, lo que significa que el acceso a la memoria desde diferentes procesadores puede tener diferentes latencias. Varios artículos en los resúmenes proporcionados discuten los desafíos y las optimizaciones relacionadas con los sistemas NUMA. Papadakis et al. proponen un mecanismo para optimizar la escalabilidad de las aplicaciones administradas en sistemas NUMA, resultando en un mejor performance [25]. Chehab et al. presentan un Compositional Lock Framework (CLoF) para sistemas NUMA multinivel, que supera las cerraduras con conocimiento de NUMA de última generación [26]. Park et al. presentan una extensión RCU (RCX) que proporciona actualizaciones de RCU

altamente escalables en sistemas NUMA, mejorando el desempeño en comparación con los enfoques existentes [27]. Memarzia et al. evalúan estrategias para acelerar cargas de trabajo de análisis de datos intensivos en memoria en sistemas NUMA, demostrando aceleraciones significativas [28]. Tripathy y Green presentan una implementación de tabla hash multi-GPU que logra un alto rendimiento, comparable a las tablas hash distribuidas [29].

Aunque la memoria de un sistema NUMA se comparte entre todos procesadores, los tiempos de acceso a las distintas partes de la memoria varían en función de la topología. Los sistemas NUMA abarcan una amplia variedad de arquitecturas de CPU, topologías y tecnologías de interconexión. Por ello, no existe una norma estándar sobre cómo debe ser la topología de un sistema NUMA. Debido a la variedad de topologías y aplicaciones NUMA, el ajuste fino del algoritmo a la configuración de una sola máquina no necesariamente logrará un rendimiento óptimo en otras máquinas. Con tiempo y recursos suficientes, las aplicaciones podrían ajustarse a las diferentes configuraciones del sistema en las que se despliegan. Sin embargo, en el mundo real, esto no siempre es factible. Por lo tanto, es deseable buscar soluciones que puedan mejorar el rendimiento en general sin ajustar el código [28].

2.2.4. Sistema UMA

El sistema UMA es un sistema binario que ha sido objeto de varios estudios. Wang et al. analizaron un binario de doble línea eclipsante en el sistema UMA utilizando fotometría TESS y observaciones espectroscópicas. Obtuvieron los parámetros atmosféricos y velocidades radiales de las estrellas componentes y determinaron sus masas y radios. También identificaron oscilaciones excitadas por las mareadas y modos de presión de bajo orden en el sistema [30]. Baumgartner estudió las propiedades físicas de un sistema tipo W UMA en el sistema UMA llamado LO Andromedae. Derivaron el período y los parámetros estelares de LO Y, incluyendo la relación de masa, inclinación, y temperaturas de las estrellas primarias y secundarias. También observaron un aumento acelerado en el periodo de LO And, que se atribuyó a la transferencia de masa y a la posible presencia de una tercera estrella [31]. Cuntz et al. examinaron la dinámica de posibles cometas alrededor de 47 UMA, un sistema conocido por albergar tres planetas tipo Júpiter. Investigaron la interacción de exocometas con los planetas y la probabilidad de colisiones

cometarias, así como el transporte de agua a planetas de masa terrestre en el sistema [30][32].

El sistema se compone esencialmente de Agente Móvil Ubicuo (UMA) para ayudar al médico monitor y de Agentes de Dispositivo (DevA) integrados en cada dispositivo. En el lado del paciente, propusimos el Agente Paciente (PatA) para asistir al paciente, recoger información de los sensores y asegurar la comunicación con el médico monitor. A continuación, detallamos cada agente Móvil. Las redes móviles ad hoc (MANET) están formadas por un conjunto de nodos móviles que no están limitados por ninguna infraestructura [33].

Ilustración 2: [33] N. Bouchemal, "Esquema de programación paralela", 2017

2.3. Paradigmas de programación paralela

Los paradigmas de programación paralela constituyen una parte fundamental en el desarrollo de software eficiente en sistemas con capacidades de procesamiento simultáneo. Entre los principales paradigmas se encuentran enfoques basados en diferentes estilos de lenguaje de programación, cada uno con sus propias ventajas y aplicaciones [34].

El paradigma de la programación lógica es uno de ellos, destacándose por su potencial en la explotación automatizada del paralelismo. Como señalan Dovier et al. en sus artículos y estudios exhaustivos [35], la programación lógica permite la representación y solución de problemas mediante un conjunto de hechos y reglas lógicas, lo que facilita la distribución y ejecución paralela de tareas. Otro paradigma importante es la programación funcional. Mencionada por Sayar y Ergün [34], esta modalidad está ganando popularidad en el ámbito de la computación paralela debido a su simplicidad y

eficacia en el diseño de software para sistemas paralelos. La programación funcional, centrada en la inmutabilidad de los datos y las funciones como ciudadanos de primera clase, es especialmente adecuada para entornos donde la consistencia de los datos y la ausencia de efectos secundarios son cruciales [34].

La programación imperativa, aunque no está explícitamente mencionada en los resúmenes referidos, es un paradigma comúnmente utilizado en la programación paralela. Este enfoque, que implica la escritura de instrucciones paso a paso para cambiar el estado del programa, se presta bien para ciertos tipos de problemas paralelos, especialmente aquellos que requieren una gestión detallada del estado y la memoria [36].

Estos paradigmas de programación se implementan utilizando varios modelos y técnicas, como comentaron Sayar y Ergün. Cada paradigma ofrece un enfoque distinto para aprovechar el paralelismo, ya sea a través de la división automática de tareas, el enfoque en la inmutabilidad de los datos, o la gestión explícita del estado del programa [37]. En conjunto, proporcionan un conjunto rico y diverso de herramientas para mejorar la eficiencia de los cálculos en entornos de computación paralela.

2.3.1. Programación secuencial vs programación paralela

2.3.1.1. Programación secuencial

La programación secuencial es el enfoque tradicional de la computación, donde las instrucciones se ejecutan una tras otra en un orden lineal. Este paradigma, predominante durante los primeros días de la informática, se basa en la ejecución paso a paso de un conjunto de instrucciones por un solo procesador [38]. En la programación secuencial, cada operación debe completarse antes de que comience la siguiente, lo que implica una estructura de programación más simple y directa, pero a menudo menos eficiente en términos de velocidad de procesamiento [39].

2.3.1.2. Programación paralela

En contraste, la programación paralela implica dividir un problema en partes que pueden ser resueltas simultáneamente. Este enfoque aprovecha la presencia de múltiples unidades de procesamiento, como en las modernas arquitecturas de computadoras multicore [1]. En la programación paralela, diferentes partes del

programa se ejecutan al mismo tiempo en diferentes procesadores, lo que puede llevar a una reducción significativa en el tiempo total de ejecución para ciertos tipos de tareas [40].

2.3.1.3. Comparación y uso

La elección entre programación secuencial y paralela depende de varios factores, incluyendo la naturaleza del problema a resolver, los recursos de hardware disponibles y los objetivos de rendimiento [41]. Mientras que la programación secuencial sigue siendo útil y preferida para tareas sencillas y lineales, la programación paralela es cada vez más relevante en la era actual, donde la velocidad de procesamiento y la eficiencia son críticas [42]. En tareas complejas, especialmente aquellas que involucran grandes conjuntos de datos o cálculos intensivos, la programación paralela puede ofrecer ventajas sustanciales [41][42] .

2.3.2. Modelos de programación paralela (Pthreads, OpenMP)

Los modelos de programación en paralelo, como Pthreads y OpenMP, se han utilizado ampliamente en la computación de alto rendimiento (HPC) [21]. Estos modelos proporcionan a los desarrolladores la posibilidad de crear versiones paralelas de sus códigos secuenciales, lo que permite mejorar el rendimiento y la escalabilidad [25]. OpenMP, en particular, se ha ampliado para admitir el paralelismo de tareas en la programación de clústeres, lo que simplifica el proceso de desarrollo y el mantenimiento. Además, se ha utilizado la combinación de los modelos de programación MPI (Message Passing Interface) y Open MultiProcessing (OpenMP) para mejorar la eficiencia del cálculo de modelos de campos gravitatorios terrestres de alta resolución [43].

2.3.2.1. Pthreads

Pthreads (POSIX Threads) es un estándar para la programación multihilo en entornos POSIX. Permite a los programadores escribir programas que ejecutan múltiples hilos de manera concurrente, una técnica fundamental en la programación paralela [41]. Este modelo proporciona un gran control sobre la creación, sincronización y manejo de hilos, lo que resulta esencial en aplicaciones donde la precisión en el control de la concurrencia es crucial. Sin embargo, su nivel de control detallado también conlleva

una mayor complejidad en el diseño y la depuración del software, lo que puede ser un desafío en aplicaciones grandes y complejas [40].

2.3.2.2. OpenMP

OpenMP (Open Multi-Processing) representa otro enfoque importante en la programación paralela. Es un conjunto de directivas de compilador, rutinas de biblioteca y variables de entorno que simplifican la creación de programas paralelos en sistemas de memoria compartida [33]. La ventaja clave de OpenMP es su relativa facilidad de uso, permitiendo a los desarrolladores paralelizar programas existentes con mínimas modificaciones al código fuente. Este modelo es ampliamente utilizado en computación de alto rendimiento debido a su capacidad para escalar eficientemente en procesadores multicore y multiproceso [40].

2.3.2.3. Comparación y selección

La elección entre Pthreads y OpenMP depende de varios factores, incluyendo el entorno de programación, la naturaleza del problema y la experiencia del programador [41]. Pthreads es más adecuado para proyectos que requieren un control fino sobre los hilos de ejecución, mientras que OpenMP es preferido en escenarios donde la facilidad de uso y la portabilidad son prioritarias [32]. Ambos modelos tienen un lugar importante en el ecosistema de la programación paralela, y su selección adecuada puede influir significativamente en el éxito y la eficiencia de un proyecto de software paralelo.

2.3.2.4. Aplicaciones y desarrollo a futuro

Los modelos de programación paralela como Pthreads y OpenMP continuarán evolucionando a medida que la computación paralela se vuelve cada vez más integral en el desarrollo de software [37]. La demanda de mayor rendimiento y eficiencia energética en aplicaciones de computación de alto rendimiento, procesamiento de datos a gran escala y sistemas de inteligencia artificial impulsará la innovación en estos modelos, así como la aparición de nuevos paradigmas y herramientas en el campo de la programación paralela [38].

2.4. Hadoop: MapReduce

2.4.1. Introducción a Hadoop y MapReduce

Hadoop es una estructura de código abierto que posibilita la creación y ejecución de aplicaciones distribuidas destinadas al procesamiento de volúmenes extensos de datos, Es un marco de trabajo empleado para la creación y ejecución de aplicaciones distribuidas, posibilitando el procesamiento de volúmenes considerables de datos. Hadoop se conforma por dos módulos: Hadoop Distributed File System (HDFS), y HadoopMapReduce [41].

La característica de ser un software de código abierto implica que sea accesible al público, descargable de forma gratuita, adaptable según las necesidades individuales y susceptible de distribución opcionalmente [32]. En los sistemas informáticos convencionales, una única base de datos alimenta a diferentes computadoras conectadas en red para sus propios fines. Esto resulta en la necesidad de que cada computadora espere a que otras finalicen sus procesos antes de poder iniciar los suyos, generando un cuello de botella en la ejecución de tareas con grandes volúmenes de datos [33].

El sistema de archivos distribuido HDFS opera mediante dos categorías de nodos: el nodo maestro, también conocido como Namenode, y los nodos esclavos, denominados Datanodes. El nodo maestro asume responsabilidades como gestionar los punteros, organizar los nodos y almacenar su ubicación. Por otro lado, los nodos esclavos únicamente se ocupan de almacena [34]. MapReduce en Hadoop se encarga de fragmentar los datos en partes más pequeñas para optimizar y acelerar su procesamiento. La función Map se encarga de realizar un mapeo, generando un par para cada dato de entrada y creando una lista de datos emparejados. Por otro lado, la función Reduce examina la lista de datos emparejados para obtener el resultado deseado, posibilitando así un trabajo en paralelo [41],[33].

La naturaleza de ser un software distribuido de Hadoop se debe a que, durante su ejecución, opera en conjunto con un conjunto de computadoras conectadas entre sí mediante una red. Es el propio software el que decide cómo distribuir la información entre estas computadoras, lo que posibilita el acceso y la manipulación de los datos desde cualquier máquina [35].

2.4.2. Aplicaciones y casos de uso de Hadoop

Hadoop se utiliza en una variedad de aplicaciones en diferentes industrias debido a su capacidad para procesar y almacenar grandes cantidades de datos de manera distribuida [39].

3. Computación distribuida

3.1. Modelos de computación distribuida

3.1.1. Arquitectura cliente-servidor

La arquitectura cliente-servidor es un diseño de sistema donde los clientes, como aplicaciones móviles o navegadores web, se comunican con un servidor central para solicitar y recibir datos o servicios. Permite una distribución eficiente de tareas y recursos, con el servidor manejando el procesamiento y almacenamiento de datos mientras que los clientes manejan la interfaz de usuario y las interacciones de los usuarios. Esta arquitectura se utiliza en diversas aplicaciones, como sistemas de pago [44], sistemas de automatización de índices bibliotecarios, y migración de comunicación cliente-servidor en empresas de TI . También juega un papel en el desarrollo de sistemas de software para la formación y simulación médica [45]. En el contexto del aprendizaje federado, la arquitectura cliente-servidor se utiliza para el aprendizaje colaborativo de modelos y la fusión en el marco de aprendizaje por refuerzo federado (FRL) fuera de política [46]

En un sistema de procesos asíncronos, donde cada proceso opera a su propia velocidad arbitraria y no determinista, se pueden identificar dos tipos de procesos: clientes y servidores. Se denomina C al conjunto de clientes y R al conjunto de servidores, ambos conjuntos pueden ser potencialmente infinitos. A cada proceso en este sistema se le asigna un protocolo específico que debe seguir [46].

3.1.2. Modelos Peer-to-Peer

Los modelos peer-to-peer (P2P) permiten el procesamiento descentralizado, el uso compartido de recursos y la comunicación sin necesidad de servidores centralizados o intermediarios. Las redes P2P han demostrado ser confiables, adaptables y tienen varias ventajas, como una mejor tolerancia a fallas, privacidad, seguridad y uso compartido simplificado de recursos [48]. En el sector energético están surgiendo mercados

energéticos P2P, lo que permite a los hogares comprar y vender electricidad renovable de manera directa. Estos mercados están influenciados por factores sociales, económicos y ambientales, y su desempeño depende de factores como la relación entre prosumidores y consumidores, las condiciones ambientales y la ubicación geográfica [48]. Existe literatura académica significativa que examina los diseños de mercado de los modelos de comercio de energía P2P, los cuales pueden ser categorizados en seis arquetipos. Sin embargo, todavía hay lagunas de evidencia que deben abordarse, como las restricciones físicas, el diseño holístico del mercado, la escalabilidad, la seguridad de la información y la privacidad de los participantes en el mercado [48]. Los servicios de acomodación P2P también se benefician de ventajas económicas y percepciones de confianza, que influyen en las actitudes de los consumidores hacia estos servicios [49]. Las soluciones técnicas factibles, incluidos los modelos de previsión energética, pueden permitir el despliegue de sistemas P2P de gestión de energía, lo que lleva a reducciones de costos y a la validación de transacciones energéticas basadas en subastas [50].

3.2. RPC y RMI

3.2.1. Concepto de llamadas a procedimientos remotos (RPC)

Las Llamadas a Procedimientos Remotos (RPC) son una técnica fundamental en la computación distribuida que permite a un programa realizar un procedimiento en otro espacio de memoria, generalmente en otro equipo, como si fuera una llamada local [51]. Este enfoque es esencial para la interacción entre sistemas en una red, proporcionando una abstracción que oculta la complejidad de la comunicación de red al programador [52]. En la práctica, RPC involucra el envío de una solicitud del cliente al servidor, que ejecuta el procedimiento y devuelve un resultado. Este mecanismo implica desafíos como la serialización de datos (convertir datos en un formato que pueda transmitirse a través de la red) y la gestión de diferentes entornos de ejecución y lenguajes de programación [53].

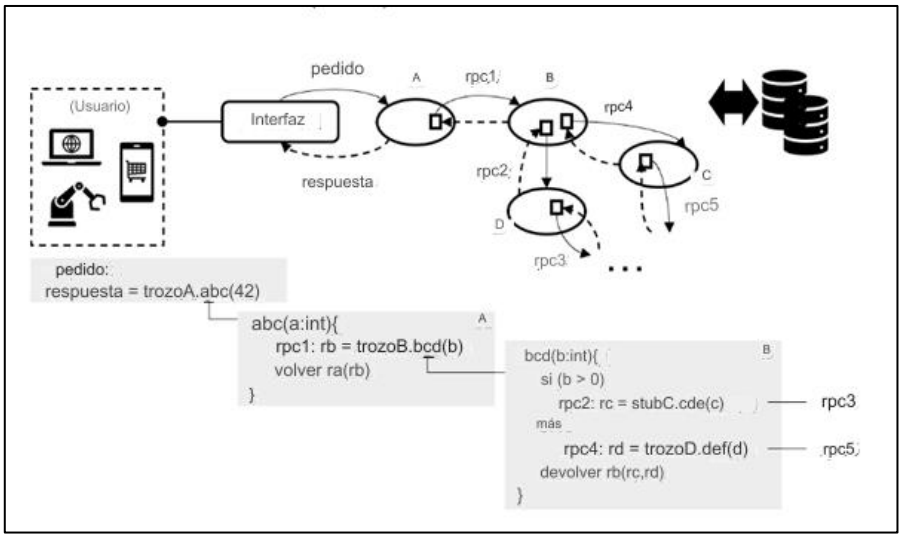

Ilustración 3: [53] M. Zhang, "API basadaen RPC de microservicios", 2023

Un aspecto clave de RPC es que se diseñó para ser independiente del lenguaje de programación, lo que permite su implementación en diversos entornos. Sin embargo, esta flexibilidad viene con la necesidad de establecer un contrato o protocolo claro, generalmente definido en un lenguaje de definición de interfaz (IDL), que especifica los tipos de datos y los formatos que se intercambian [53]. RPC ha sido fundamental en el desarrollo de aplicaciones distribuidas, especialmente en entornos donde se requiere una fuerte coherencia y fiabilidad en la comunicación entre diferentes componentes del sistema.

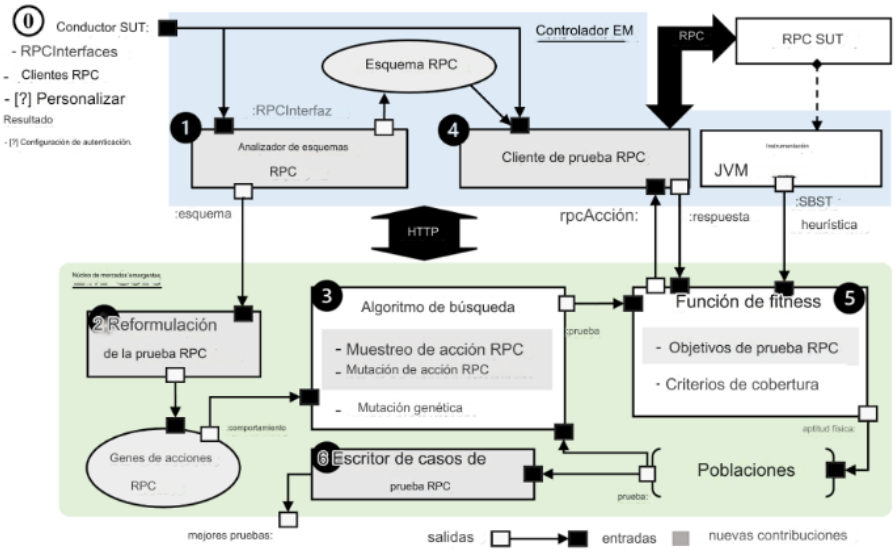

Ilustración 4: [53] M. Zhang, "Descripción general del enfoque RPC", 2023

3.2.2. Llamadas de métodos remotos (RMI) en java

Las Llamadas de Métodos Remotos (RMI) en Java representan una implementación específica de RPC que se ajusta al paradigma de programación orientada a objetos de Java. RMI permite que un objeto en una Java Virtual Machine (JVM) invoque métodos de un objeto en otra JVM, potencialmente en diferentes máquinas [51]. Este modelo simplifica enormemente el desarrollo de aplicaciones distribuidas en Java, ya que maneja los detalles de la comunicación de red, permitiendo a los desarrolladores centrarse en la lógica de negocio [45].

RMI se integra perfectamente con las características estándar de Java, como la gestión de objetos y la recolección de basura, lo que lo hace particularmente robusto y fácil de usar en el ecosistema de Java. Un desafío con RMI es que está limitado a entornos de Java, lo que significa que no es adecuado para sistemas distribuidos que involucran múltiples lenguajes de programación o plataformas [46]. Sin embargo, dentro del mundo de Java, RMI es una herramienta poderosa para la creación de aplicaciones distribuidas, como sistemas de bases de datos distribuidas, aplicaciones web y servicios en la nube. La naturaleza orientada a objetos de RMI facilita la creación de sistemas distribuidos complejos y modulares, donde los objetos remotos pueden interactuar con una interfaz común, independientemente de su ubicación física [47].

3.3. Ventajas y desventajas

3.3.1. Ventajas de la computación distribuida en términos de escalabilidad y redundancia.

Este es un ejemplo el cual se analiza los problemas de redundancia en la computación de matrices distribuidas que son comunes en el análisis de datos a gran escala, Se presentan métodos automáticos y adaptativos para eliminar subexpresiones redundantes en consultas de álgebra lineal, especialmente en circuitos de multiplicación de matrices, la implementación experimental de ReMac en SystemDS demuestra eficiencia y asequibilidad en comparación con las soluciones actuales[51].

Se gestiona la redundancia de datos en entornos de alta disponibilidad, especialmente servidores de control de sesiones en redes de operadores. Utiliza codificadores automáticos, un algoritmo de reducción de dimensionalidad basado en redes

neuronales, para detectar anomalías en las redes, Con el uso de hash consistente, este método se diseña para evitar la pérdida de datos en caso de problemas graves con equipos de red, como servidores y enrutadores [52].

La estrategia de almacenamiento descentralizado basada en la estimación de la topología de gráficos para optimizar la redundancia en redes propensas a fallas, se caracteriza por una mayor robustez y longevidad, y proporciona un aumento significativo de la eficiencia al reducir la transmisión y el almacenamiento de datos, reduciendo así el consumo energético [43]

3.3.2. Desventajas, como la complejidad de administración y la comunicación interprocesos.

Este paradigma abordará cuestiones relacionadas con la computación distribuida, como el manejo de la complejidad y la comunicación entre procesos. Aunque no se considera directamente una desventaja, destaca la necesidad de protocolos eficientes en todas las etapas del ciclo de vida de una red peer-to-peer (P2P), presentando una respuesta a los problemas comunes en los sistemas descentralizados. Centrarse en superar las limitaciones de información en un solo nodo y desarrollar arquitecturas P2P mejoradas ofrece soluciones a la complejidad de los procesos y los problemas de comunicación [51].

Los clientes que buscan precios más bajos suelen elegir proveedores menos capaces, lo que agrava el impacto negativo en los resultados del proyecto, además, esto se agrava por la disminución de la competencia del proveedor, de modo que elegir uno con un historial de errores del 15 % aumenta la tasa de fallas en un 33 % frente a uno con un historial de errores del 5 %, y los clientes pueden reducir significativamente la probabilidad de fracaso del proyecto al disminuir el énfasis en los precios bajos al seleccionar un proveedor [52].

3.4. Kubernetes

3.4.1. Introducción a Kubernetes y orquestación de contenedores.

Kubernetes, también conocido como K8s, es un sistema de código abierto diseñado para automatizar la implementación, gestión y escalabilidad de aplicaciones contenerizadas en clústeres distribuidos. Representa la evolución del proyecto Borg de Google, que

contaba con más de una década de experiencia ejecutando aplicaciones escalables en entornos de producción. Fue liberado a la comunidad de código abierto en 2014 y, desde entonces, no ha dejado de crecer [53].

Kubernetes automatiza la distribución y programación más eficiente de aplicaciones contenerizadas en los nodos del clúster (físicos o virtuales), equilibrando el uso de recursos en cada nodo. Estos componentes también pueden estar integrados en K8s (cabe destacar que no todos necesitan implementarse en un orquestador). Es importante señalar que los administradores y usuarios de K8s pueden configurar una amplia gama de componentes para controlar mejor el uso de recursos y el rendimiento final de la aplicación [53].

El proceso general de gestión de recursos en K8s se puede resumir de la siguiente manera: Cuando un usuario solicita la creación de un pod con ciertos recursos informáticos, el nodo maestro de K8s recibe la solicitud. La solicitud se envía al servidor

En general, las características de las aplicaciones que se ejecutarán en clústeres controlados por Kubernetes afectan la programación, ya que cada aplicación tiene diferentes requisitos de calidad de servicio (QoS) que deben cumplirse. En otras técnicas se presentan de manera genérica [50].

Las métricas de evaluación están relacionadas con el objetivo que deben optimizar los algoritmos de programación. En algunos documentos, la evaluación del rendimiento se centra en una sola métrica, pero la mayoría examina varias métricas. Cuantas más métricas se utilicen en la evaluación del rendimiento, más complejo será el proceso de toma de decisiones. Por lo tanto, generalmente se establecen diferentes compensaciones para equilibrar la eficiencia del algoritmo de programación propuesto y la calidad final de la solución generada [52].

3.4.2. Implementación de aplicaciones en clústeres y Kubernetes

Clústeres

El Análisis de Clusters, también conocido como Análisis de conglomerados, es una técnica dentro del Análisis Exploratorio de Datos diseñada para abordar problemas de clasificación [50]. Su objetivo principal es organizar objetos (personas, cosas, animales, plantas, variables, etc.) en grupos, denominados conglomerados o clusters, de manera

que la similitud entre los miembros de un mismo cluster sea más pronunciada que la similitud entre los miembros de diferentes clusters. Cada cluster se caracteriza y describe como la clase a la que pertenecen sus miembros [51].

4. Practica

4.1. RMI

Problema: Sistema de Cálculo Matemático Distribuido

Descripción: Considere una situación en la que se necesita realizar operaciones matemáticas básicas (suma, resta, multiplicación y división) en un entorno distribuido, donde diferentes usuarios pueden requerir cálculos simultáneos desde múltiples ubicaciones. Este escenario es común en contextos educativos, financieros o científicos, donde los cálculos se necesitan en tiempo real y deben ser precisos y confiables. La solución a este problema implica el desarrollo de un Sistema de Cálculo Matemático Distribuido (SCMD) que permita realizar operaciones matemáticas de manera eficiente y distribuida.

La solución propuesta utiliza la tecnología de Java RMI (Remote Method Invocation) para crear un sistema cliente-servidor. Los usuarios, actuando como clientes, pueden enviar peticiones de cálculos matemáticos al servidor, el cual procesa estas solicitudes y devuelve los resultados. El sistema consta de cuatro componentes principales:

1. **Cliente:** Una interfaz de usuario que permite a los usuarios ingresar los números y seleccionar la operación matemática deseada. Este cliente comunica las solicitudes al servidor y muestra los resultados obtenidos.
2. **Interfaz Remota:** Define los métodos disponibles para las operaciones matemáticas (suma, resta, multiplicación, división) que pueden ser invocados remotamente por los clientes.
3. **MainServer:** Configura el entorno del servidor, registrando los servicios ofrecidos y esperando las peticiones de los clientes.
4. **Server Implements:** Implementa los métodos definidos en la Interfaz Remota y contiene la lógica para realizar las operaciones matemáticas.

Pasos

1. Crear la interfaz remota

   Definir un contrato común entre el cliente y el servidor. Al especificar los métodos que pueden ser llamados remotamente, se establece una base para la comunicación y el intercambio de datos. Esta interfaz es esencial para asegurar que tanto el cliente como el servidor entiendan y acuerden la forma y el tipo de operaciones que se pueden realizar.

   Codigo:

```
package RMI;

import java.rmi.Remote;

public interface RemoteInterface extends Remote {

    public int suma(int x, int y) throws Exception;

    public int resta(int x, int y) throws Exception;

    public int multiplica(int x, int y) throws Exception;

    public int divide(int x, int y) throws Exception;

}
```

2. Implementar el servidor

   2.1. ServerImplements

   Aquí se van a Realizar las operaciones matemáticas reales. Este componente actúa como el núcleo de procesamiento del sistema, ejecutando los cálculos solicitados por los clientes.

```
package Server;
import RMI.RemoteInterface;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class ServerImplements extends UnicastRemoteObject implements RemoteInterface{
    public ServerImplements() throws RemoteException{
        super();
    }
    @Override
    public int suma(int x, int y){
        return(x+y);
    }
    public int resta(int x, int y){
        return(x-y);
    }
    public int multiplica(int x, int y){
        return(x*y);
    }
    public int divide(int x, int y){
        return(x/y);
    }
}
```

2.2. Server

Aquí nos encargamos de iniciar y gestionar el servicio del servidor RMI. Este paso es crucial para que el servidor esté disponible y escuchando las solicitudes de los clientes. Al registrar el servicio y asociarlo con un nombre específico, se facilita a los clientes la localización y el uso de los servicios ofrecidos.

```
package Server;


import java.rmi.Remote;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class MainServer {
    public static void main (String args[]){
        try{
            Registry miRegistry=LocateRegistry.createRegistry(1234);
            miRegistry.rebind("Ejemplo Matematicas", (Remote) new ServerImplements());
            System.out.println("Servidor ON");
        }
        catch(Exception e){
            System.out.println(e.getMessage());
        }
    }
}
```

3. Crear clientes

Aquí vamos a proporcionar una interfaz para que los usuarios interactúen con el sistema. El cliente es responsable de recoger las entradas del usuario, enviarlas al servidor para su procesamiento y mostrar los resultados de vuelta al usuario. Esta parte del sistema hace que el proceso de cálculo sea accesible y fácil de usar para los usuarios finales.

```
package Client;
import RMI.RemoteInterface;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import javax.swing.JOptionPane;
public class Cliente {
    public static void main (String args[]){
        try{
            String valora= JOptionPane.showInputDialog("Ingrese el valor del numero uno");
            String valorb= JOptionPane.showInputDialog("Ingrese el valor del numero dos");
            int a=Integer.parseInt(valora);
            int b=Integer.parseInt(valorb);
            Registry miRegistro=LocateRegistry.getRegistry("localhost",1234);
            RemoteInterface s=(RemoteInterface) miRegistro.lookup("Ejemplo Matematicas");
            JOptionPane.showMessageDialog(null, "Resultado suma: "+s.suma(a,b));
            JOptionPane.showMessageDialog(null, "Resultado resta: "+s.resta(a,b));
            JOptionPane.showMessageDialog(null, "Resultado Multiplica: "+s.multiplica(a,b));
            JOptionPane.showMessageDialog(null, "Resultado Divide: "+s.divide(a,b));
        }catch (Exception e){
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Resultados

1. Ejecución del servidor

Una vez que se ejecute el servidor, nos va a mostrar un mensaje el cual indicara el estado del servidor, es crucial iniciar primero el servidor para que el cliente pueda hacer uso de los servicios.

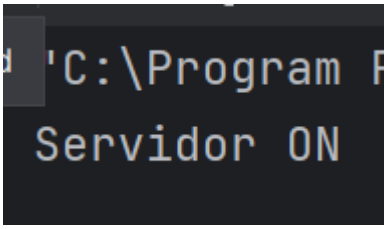

Ilustración 5: Ejecución del Servidor, Vilcacundo Jordy

2. Ejecución del cliente

Una vez que el servidor este encendido, el cliente podrá hacer uso de sus servicios, o las funciones que tenga el servidor establecido.

Ahora cuando se ejecute el cliente nos mostrara las ventanas donde tendremos que ingresar la información:

2.1. Ingrese primer número

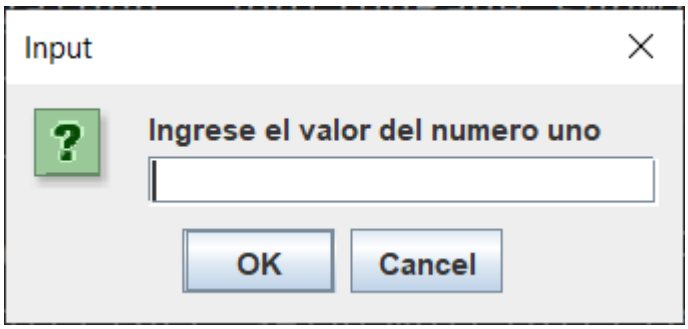

Ilustración 6: Ingreso primero número, Vilcacundo Jordy

2.2. Ingrese segundo número

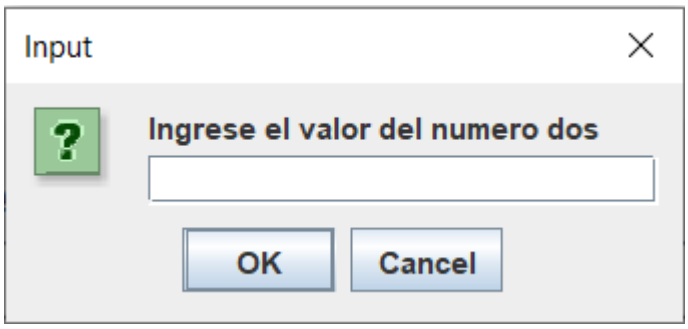

Ilustración 7: Ingreso segundo número, Vilcacundo Jordy

Una vez que ingresamos los valores, de forma transparente se realizaran los procedimientos matemáticos que fueron establecidos en los implementos del servidor.

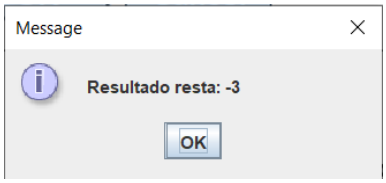

Ilustración 8: Resultado resta, Vilcacundo Jordy

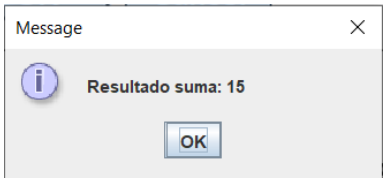

Ilustración 9: Resultado suma, Vilcacundo Jordy

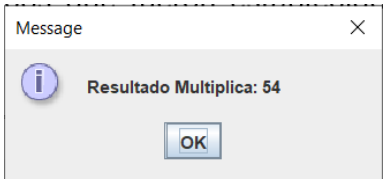

Ilustración 10: Resultado multiplicación, Vilcacundo Jordy

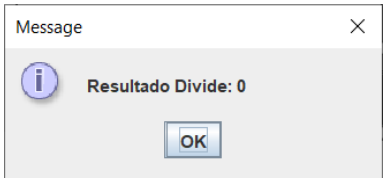

Ilustración 11: Resultado división, Vilcacundo Jordy

Los resultados mostrados son la resolución de los dos números ingresados mediante cada una de las operaciones matemáticas, el valor de división retorna 0 debido a que se está trabajando con enteros y no decimales.

4.2 RPC

Problema: Se desea implementar un Xml Cliente-Servidor en modelo RPC, usando Python, permitiendo calcular las 4 operaciones básicas.

Descripción: Se busca desarrollar un sistema Cliente-Servidor basado en XML y Modelo RPC con Python para facilitar operaciones aritméticas básicas. Este sistema permitirá a los clientes enviar solicitudes codificadas en XML al servidor, donde se procesarán las operaciones de suma, resta, multiplicación y división, retornando los resultados correspondientes. La elección de XML garantiza un intercambio de datos estructurado y legible, mientras que el Modelo RPC simplifica la comunicación entre los componentes.

Con Python como lenguaje de implementación, se espera una solución eficiente y escalable, con potencial para futuras extensiones y mejoras.

Pasos:

1. Creación del servidor XMLRPC (servidor.py), importación de modulo y configuración de funciones

```
from xmlrpc.server import SimpleXMLRPCServer

def suma(num1, num2):
    return num1 + num2

def resta(num1, num2):
    return num1 - num2

def multiplicacion(num1, num2):
    return num1 * num2

def division(num1, num2):
    return num1 / num2

def potencia(num1,num2):
    return num1 ** num2

server = SimpleXMLRPCServer(("localhost", 5048))
```

2. Registro de funciones en el servidor

```
server.register_function(suma, "suma")
server.register_function(resta, "resta")
server.register_function(multiplicacion, "multiplicacion")
server.register_function(division, "division")
server.register_function(potencia, "potencia")
```

```
print("Servidor en ejecución en http://localhost:6789")
server.serve_forever()
```

3. Creación del cliente XMLRPC (cliente.py), importación de modulo y creación del proxy servidor

```
import xmlrpc.client

def main():
    proxy = xmlrpc.client.ServerProxy("http://localhost:5048/")
```

4. Declaración de números y llamadas a los métodos del servidor

```
    try:
        resultado_suma = proxy.suma(primer_numero, segundo_numero)
        print(f"El resultado de la suma es: {resultado_suma}")

        resultado_resta = proxy.resta(primer_numero, segundo_numero)
        print(f"El resultado de la resta es: {resultado_resta}")

            resultado_multiplicacion = proxy.multiplicacion(primer_numero,
            segundo_numero)
            print(f"El resultado de la multiplicación es: {resultado_multiplicacion}")

            resultado_division = proxy.division(primer_numero, segundo_numero)
        print(f"El resultado de la división es: {resultado_division}")

            resultado_potencia = proxy.potencia(primer_numero, segundo_numero)
        print(f"El resultado de la potencia es: {resultado_potencia}")

    except ConnectionRefusedError:
```

```
        print("Error: No se pudo conectar al servidor.")
```

5. Iniciar la ejecución del programa

```
if __name__ == "__main__":
    main()
```

6. Ejecución del servidor.py

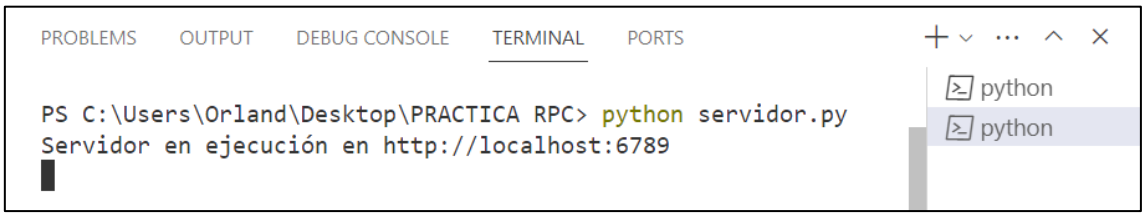

Ilustración 12: Ejecución del servidor.py, Cedeño Orlando

7. Ejecución del cliente.py

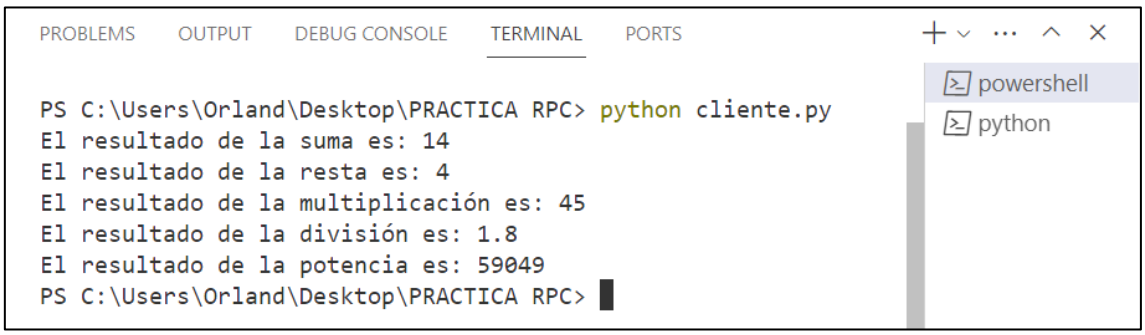

Ilustración 13: Ejecución del cliente.py, Cedeño Orlando

5. Conclusión

El análisis de la computación paralela y distribuida subraya su importancia en el avance tecnológico. Las arquitecturas paralelas como SIMD y MIMD, y modelos como Pthreads y OpenMP, mejoran significativamente el rendimiento de los sistemas informáticos. Paralelamente, la computación distribuida, a través de modelos como cliente-servidor y tecnologías como RPC, RMI y Kubernetes, optimiza la interconexión y colaboración en redes, enfrentando desafíos de escalabilidad y gestión.

Estos campos crean un ecosistema dinámico esencial para resolver desafíos computacionales complejos y responder a las demandas de la industria y la investigación. Su comprensión es vital para profesionales en ciencias de la computación y para quienes desarrollan tecnologías avanzadas, prometiendo una continua evolución e innovación en la tecnología de información y comunicación.

6. Link del Github

https://github.com/OrlandCede20/PRACT_COMPUPAR_DISTR.git

7. Referencias

[1] Y. Sitsylitsyn, "A systematic review of the literature on methods and technologies for teaching parallel and distributed computing in universities," *Ukr. J. Educ. Stud. Inf. Technol.*, vol. 11, no. 2, pp. 111–121, 2023, doi: 10.32919/uesit.2023.02.04.

[2] K. Xu and E. Darve, "ADCME MPI: Distributed machine learning for computational engineering," *CEUR Workshop Proc.*, vol. 2964, pp. 1–24, 2021.

[3] H. Shen, H. Tian, and Y. Sang, "Guest Editorial – Parallel and Distributed Computing and Applications," *Comput. Sci. Inf. Syst.*, vol. 20, no. 1, p. ix, 2023, doi: 10.2298/CSIS230100ixS.

[4] D. Ávila, A. Papavasiliou, and N. Löhndorf, "Parallel and distributed computing for stochastic dual dynamic programming," *Comput. Manag. Sci.*, vol. 19, no. 2, pp. 199–226, 2022, doi: 10.1007/s10287-021-00411-x.

[5] N. Harrison *et al.*, "A novel inverse algorithm to solve IPO-IMPT of proton FLASH therapy with sparse filters," *Int. J. Radiat. Oncol.*, pp. 1–18, 2023, doi: 10.1016/j.ijrobp.2023.11.061.

[6] H. Shukur *et al.*, "A State of Art Survey for Concurrent Computation and Clustering of Parallel Computing for Distributed Systems," *J. Appl. Sci. Technol. Trends*, vol. 1, no. 4, pp. 148–154, 2020, doi: 10.38094/jastt1466.

[7] G. Rao, J. Chen, J. Yik, and X. Qian, "SparseCore: Stream ISA and Processor Specialization for Sparse Computation," *Int. Conf. Archit. Support Program. Lang. Oper. Syst. - ASPLOS*, pp. 186–199, 2022, doi: 10.1145/3503222.3507705.

[8] L. Chang, “Cognitive Data-Centric Systems,” pp. 1–1, May 2017, doi: 10.1145/3060403.3060491.

[9] S. Gerbino, A. Lanzotti, M. Martorelli, R. Mirálbes Buil, C. Rizzi, and L. Roucoules, Eds., “Advances on Mechanics, Design Engineering and Manufacturing IV,” 2023, doi: 10.1007/978-3-031-15928-2.

[10] R. Pierrard, L. Cabaret, J. P. Poli, and C. Hudelot, “SIMD-based exact parallel fuzzy dilation operator for fast computing of fuzzy spatial relations,” *WPMVP 2020 - Proc. 2020 6th Work. Program. Model. SIMD/Vector Process. co-located with PPoPP 2020*, 2020, doi: 10.1145/3380479.3380482.

[11] T. Plano and J. Buhler, “Scheduling irregular dataflow pipelines on SIMD architectures,” *WPMVP 2020 - Proc. 2020 6th Work. Program. Model. SIMD/Vector Process. co-located with PPoPP 2020*, 2020, doi: 10.1145/3380479.3380480.

[12] D. Gerzhoy, X. Sun, M. Zuzak, and D. Yeung, “Nested MIMD-SIMD parallelization for heterogeneous microprocessors,” *ACM Trans. Archit. Code Optim.*, vol. 16, no. 4, 2019, doi: 10.1145/3368304.

[13] M. Zarubin, P. Damme, A. Krause, D. Habich, and W. Lehner, “SIMD-MIMD cocktail in a hybrid memory glass: Shaken, not stirred,” *SYSTOR 2021 - Proc. 14th ACM Int. Conf. Syst. Storage*, 2021, doi: 10.1145/3456727.3463782.

[14] V. Svjatnij, A. Liubymov, O. Miroshkin, and V. Kushnarenko, “MIMD-Simulators Based on Parallel Simulation Language,” *Lect. Notes Inst. Comput. Sci. Soc. Telecommun. Eng. LNICST*, vol. 436 LNICST, pp. 189–200, 2022, doi: 10.1007/978-3-031-01984-5_16/COVER.

[15] M. B. S. Ahmad and A. Cheung, “Optimizing data-intensive applications automatically by leveraging parallel data processing frameworks,” *Proc. ACM SIGMOD Int. Conf. Manag. Data*, vol. Part F1277, pp. 1675–1678, 2017, doi: 10.1145/3035918.3056440.

[16] H. Funke and J. Teubner, “Data-parallel query processing on non-uniform data,” *Proc. VLDB Endow.*, vol. 13, no. 6, pp. 884–897, Feb. 2020, doi:

10.14778/3380750.3380758.

[17] L. W. Remedios *et al.*, “Exploring shared memory architectures for end-to-end gigapixel deep learning,” *arXiv*, pp. 4–7, 2023, doi: 10.48550/arXiv.2304.12149.

[18] L. Cudennec, “Software-Distributed Shared Memory for Heterogeneous Machines: Design and Use Considerations Loïc Cudennec. Software-Distributed Shared Memory for Heterogeneous Machines: Design and Use Considerations. [Research Report] Commissariat à l’Energie Atomique e,” 2020, [Online]. Available: https://hal.archives-ouvertes.fr/hal-02928168

[19] F. Lopez, E. Chow, S. Tomov, and J. Dongarra, “Asynchronous SGD for DNN training on shared-memory parallel architectures,” *Proc. - 2020 IEEE 34th Int. Parallel Distrib. Process. Symp. Work. IPDPSW 2020*, pp. 995–998, 2020, doi: 10.1109/IPDPSW50202.2020.00168.

[20] C. Castes, E. Agullo, O. Aumage, and E. Saillard, “Decentralized in-order execution of a sequential task-based code for shared-memory architectures,” *Proc. - 2022 IEEE 36th Int. Parallel Distrib. Process. Symp. Work. IPDPSW 2022*, pp. 552–561, 2022, doi: 10.1109/IPDPSW55747.2022.00095.

[21] A. Puri and J. Jose, “Design , Modeling , and Analysis of Memory Allocation Policies for Rack-Scale Memory Disaggregation Design , Modeling , and Analysis of Memory,” pp. 0–29, 2023.

[22] M. Eren, E. Skau, P. Romero, and S. Eidenbenz, “Distributed Out-of-Memory NMF on CPU / GPU Architectures,” pp. 0–28, 2023.

[23] K. Keeton *et al.*, “MODC: Resilience for disaggregated memory architectures using task-based programming,” 2021, [Online]. Available: http://arxiv.org/abs/2109.05329

[24] I. Boureima, M. Bhattarai, M. E. Eren, N. Solovyev, H. Djidjev, and B. S. Alexandrov, “Distributed Out-of-Memory SVD on CPU/GPU Architectures,” *2022 IEEE High Perform. Extrem. Comput. Conf. HPEC 2022*, 2022, doi: 10.1109/HPEC55821.2022.9926288.

[25] O. Papadakis *et al.*, “Scaling Up Performance of Managed Applications on NUMA Systems,” *Int. Symp. Mem. Manag. ISMM*, pp. 1–14, 2023, doi: 10.1145/3591195.3595270.

[26] R. L. De Lima Chehab, A. Paolillo, D. Behrens, M. Fu, H. Härtig, and H. Chen, “CLoF: A Compositional Lock Framework for Multi-level NUMA Systems,” *SOSP 2021 - Proc. 28th ACM Symp. Oper. Syst. Princ.*, vol. 2, pp. 851–865, 2021, doi: 10.1145/3477132.3483557.

[27] S. Park, P. E. McKenney, L. Dufour, and H. Y. Yeom, “An HTM-based update-side synchronization for RCU on NUMA systems,” *Proc. 15th Eur. Conf. Comput. Syst. EuroSys 2020*, 2020, doi: 10.1145/3342195.3387527.

[28] P. Memarzia, S. Ray, and V. C. Bhavsar, “Toward Efficient In-memory Data Analytics on NUMA Systems,” 2019, [Online]. Available: http://arxiv.org/abs/1908.01860

[29] A. Tripathy and O. Green, “Scalable Hash Table for NUMA Systems,” 2021, [Online]. Available: http://arxiv.org/abs/2104.00792

[30] K. Wang, A. Ren, M. F. Andersen, F. Grundahl, T. Chen, and P. L. Pallé, “FX UMa: A New Heartbeat Binary System with Linear and Nonlinear Tidal Oscillations and δ Sct Pulsations,” *Astron. J.*, vol. 166, no. 2, p. 42, 2023, doi: 10.3847/1538-3881/acdac9.

[31] P. Peng, “A new investigation on the characteristics of a W UMa type system LO Andromedae,” 2023.

[32] M. Cuntz, B. Loibnegger, and R. Dvorak, “Exocomets in the 47 UMa System: Theoretical Simulations Including Water Transport,” *Astron. J.*, vol. 156, no. 6, p. 290, 2018, doi: 10.3847/1538-3881/aaeac7.

[33] N. Bouchemal, R. Maamri, and N. Bouchemal, “UMA : Ubiquitous Mobile Agent System for Healthcare Telemonitoring,” 2017.

[34] A. Dovier, A. Formisano, G. Gupta, M. V. Hermenegildo, E. Pontelli, and R. Rocha, “Parallel Logic Programming: A Sequel,” *Theory Pract. Log. Program.*, vol. 22, no. March, pp. 905–973, 2022, doi: 10.1017/S1471068422000059.

[35] A. Ruokamo, “Parallel computing and parallel programming models : application in digital image processing on mobile systems and personal mobile devices,” *Univ. Oulu Dep. Inf. Process. Sci.*, pp. 1–36, 2018.

[36] X. Wang, B. Shi, and Y. Fang, “Distributed Systems for Emerging Computing: Platform and Application,” *Future Internet*, vol. 15, no. 4. MDPI, Apr. 01, 2023. doi: 10.3390/fi15040151.

[37] P. Kochovski, R. Sakellariou, M. Bajec, P. Drobintsev, and V. Stankovski, “An architecture and stochastic method for database container placement in the edge-fog-cloud continuum,” in *Proceedings - 2019 IEEE 33rd International Parallel and Distributed Processing Symposium, IPDPS 2019*, Institute of Electrical and Electronics Engineers Inc., May 2019, pp. 396–405. doi: 10.1109/IPDPS.2019.00050.

[38] A. Grothey and K. McKinnon, “On the effectiveness of sequential linear programming for the pooling problem,” *Ann. Oper. Res.*, vol. 322, no. 2, pp. 691–711, 2023, doi: 10.1007/s10479-022-05156-7.

[39] T. Cunis and B. Legat, “Sequential sum-of-squares programming for analysis of nonlinear systems,” *Proc. Am. Control Conf.*, vol. 2023-May, pp. 756–762, 2023, doi: 10.23919/ACC55779.2023.10156153.

[40] P. N. Telegin, A. V. Baranov, B. M. Shabanov, and A. I. Tikhomirov, “Parallelism Detection Using Graph Labelling,” *Lobachevskii J. Math.*, vol. 43, no. 10, pp. 2893–2900, 2022, doi: 10.1134/S199508022213042X.

[41] T. Sequential and R. Parallel, “Edinburgh Research Explorer Think Sequential , Run Parallel,” 2018.

[42] D. Bozidar and T. Dobravec, “Comparison of parallel sorting algorithms,” no. November, 2015, [Online]. Available: http://arxiv.org/abs/1511.03404

[43] R. Sahl, P. Dupont, C. Messager, M. Honnorat, and T. Vu La, “High-Resolution Ocean Winds: Hybrid-Cloud Infrastructure for Satellite Imagery Processing,” *IEEE Int. Conf. Cloud Comput. CLOUD*, vol. 2018-July, pp. 883–886, 2018, doi: 10.1109/CLOUD.2018.00127.

[44] M. Striani, “NRTS: A Client-Server architecture for supporting data recording, transmission and evaluation of multidisciplinary teams during the neonatal resuscitation simulation scenario,” 2023, [Online]. Available: http://arxiv.org/abs/2304.09860

[45] G. Darbord *et al.*, “Migrating the Communication Protocol of Client-Server Applications,” *IEEE Softw.*, vol. 40, no. 4, pp. 11–18, 2023, doi: 10.1109/MS.2023.3263019.

[46] S. Ferretti and G. D’Angelo, “Client/Server Gaming Architectures,” in *Encyclopedia of Computer Graphics and Games*, 2018. doi: 10.1007/978-3-319-08234-9_272-1.

[47] M. Camaioni, R. Guerraoui, J. Komatovic, M. Monti, and M. Vidigueira, “Carbon: An Asynchronous Voting-Based Payment System for a Client-Server Architecture,” *Proc.*, vol. 1, no. 1, 2022, [Online]. Available: http://arxiv.org/abs/2209.09580

[48] J. Monroe, E. R. Bolton, and E. Z. Berglund, “Evaluating Peer-to-Peer Electricity Markets across the U.S. Using an Agent-Based Modeling Approach,” *Adv. Environ. Eng. Res.*, vol. 04, no. 01, pp. 1–40, 2023, doi: 10.21926/aeer.2301017.

[49] M. Ratilla, S. K. Dey, and M. Chovancova, “Attitude Towards Peer-to-Peer Accommodation: Evidence from Tourists in the Philippines,” *Tourism*, vol. 71, no. 2, pp. 301–315, 2023, doi: 10.37741/t.71.2.5.

[50] Z. Vale, “Sharing Market,” 2022.

[51] J. Chen *et al.*, “Remote Procedure Call as a Managed System Service,” *Proc. 20th USENIX Symp. Networked Syst. Des. Implementation, NSDI 2023*, pp. 141–159, 2023.

[52] B. Zhao, W. Wu, and W. Xu, “NetRPC: Enabling In-Network Computation in Remote Procedure Calls,” *Proc. 20th USENIX Symp. Networked Syst. Des. Implementation, NSDI 2023*, no. Idl, pp. 199–217, 2023.

[53] M. Zhang, A. Arcuri, Y. Li, Y. Liu, and K. Xue, “White-Box Fuzzing RPC-Based APIs with EvoMaster: An Industrial Case Study,” *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 5, pp. 1–38, 2023, doi: 10.1145/3585009.