

Arquitecturas distribuidas, Computación paralela y distribuida

Autores: Karla Almea¹, Ericksson Estévez¹, Elvis Gonzales¹, Jorge Gualpa¹

¹ Facultad de Ciencias de la Ingeniería – Universidad Técnica Estatal de Quevedo (UTEQ) -
Quevedo – Los Ríos – Ecuador
[kalmeav, ericksson.estevez2016, elvis.gonzales2016,
jorge.gualpa2015]@uteq.edu.ec

Resumen. Este trabajo explora las arquitecturas distribuidas y la computación paralela y distribuida. En primer lugar, se describen los conceptos básicos de las arquitecturas de aplicaciones y las arquitecturas web, seguido por una exploración de las arquitecturas distribuidas. Posteriormente, se aborda la computación paralela y distribuida, comenzando por discutir las arquitecturas según instrucciones y datos y las arquitecturas según la distribución de memoria. Se analizan el paradigma de programación paralela, Hadoop MapReduce y los modelos de computación distribuida. Se describen los conceptos de RPC y RMI, y se discuten sus ventajas y desventajas. Además, se presenta Kubernetes, una plataforma de orquestación de contenedores que se utiliza para implementar y gestionar aplicaciones en entornos de computación distribuida. Finalmente, se documentan las prácticas realizadas en clase sobre Hilos y RMI. En conjunto, este informe proporciona una visión general de estos temas, que puede ser utilizada como punto de partida para una exploración más profunda de cada uno de ellos.

Palabras claves: Arquitecturas distribuidas, Computación paralela, RCP, RMI.

1 Introducción

En la actualidad, el procesamiento de grandes cantidades de datos y la necesidad de realizar cálculos complejos en tiempo real se han convertido en aspectos fundamentales en diferentes áreas del conocimiento. Para satisfacer estas necesidades, se han desarrollado tecnologías como las arquitecturas distribuidas y la computación paralela y distribuida [1].

La arquitectura distribuida se refiere a la estructura de un sistema de computación en el que los componentes se distribuyen entre varias máquinas, que trabajan juntas para proporcionar un servicio o resolver un problema. Esto se logra mediante la interconexión de computadoras y la coordinación de sus recursos [2].

Por otro lado, la computación paralela y distribuida se refiere a la capacidad de un sistema para realizar múltiples tareas simultáneamente a través de la utilización de múltiples procesadores. La computación paralela se centra en la división de una tarea

en partes más pequeñas que se ejecutan en paralelo, mientras que la distribución implica la ejecución de tareas en diferentes máquinas interconectadas [3].

La ventaja de la computación paralela y distribuida es que permite acelerar la solución de problemas complejos y aumentar la eficiencia en la utilización de recursos informáticos. Esto se logra dividiendo un problema en partes más pequeñas que se pueden resolver de forma independiente y simultánea. La combinación de los resultados de estas partes luego permite obtener la solución final [4].

En este trabajo se abordarán los temas Arquitecturas distribuidas y Computación paralela y distribuida, en donde se explicará conceptos básicos de cada uno

2 Arquitecturas distribuidas

La arquitectura distribuida es una de las alternativas más comunes para solucionar los inconvenientes que surgen en los sistemas de software modernos. Al emplear esta arquitectura, diversas partes de una aplicación pueden ejecutarse en múltiples dispositivos de red, lo que simplifica la gestión de grandes volúmenes de datos y contribuye a un sistema que es altamente tolerante a fallas y siempre disponible. Para crear arquitecturas distribuidas, se han desarrollado varios enfoques y tecnologías, que incluyen arquitecturas cliente-servidor, microservicios y arquitecturas de contenedores. Estos enfoques están diseñados para abordar problemas relacionados con la escalabilidad, la disponibilidad y la tolerancia a fallas [2].

La arquitectura distribuida se puede crear de varias maneras, siendo una de las más utilizadas la arquitectura cliente-servidor. En esta estructura, los clientes se comunican con un servidor central para solicitar y recibir datos. Además de esta, otra arquitectura distribuida bastante difundida es la arquitectura peer-to-peer (P2P), en la que los dispositivos se comunican directamente entre sí [5].

La creación de una arquitectura distribuida requiere el uso de herramientas y tecnologías específicas, como middleware, sistemas de administración de bases de datos distribuidas y sistemas de almacenamiento distribuido [6]. Sin embargo, antes de seguir con las herramientas y tecnologías se procederá a explicar qué es Arquitectura de aplicaciones.

2.1 Arquitecturas de Aplicaciones

¿Qué es una arquitectura de aplicación?

La arquitectura de aplicaciones es el diseño y la estructura de una aplicación de software, es decir, es un diagrama estructural que describe cómo las aplicaciones de software de una organización se unen y se comunican entre sí para satisfacer las necesidades de negocio o del cliente. La arquitectura de aplicaciones es fundamental para garantizar la escalabilidad y confiabilidad de las aplicaciones, y ayuda a las empresas a detectar posibles problemas en la funcionalidad [7].

Existen diferentes tipos de arquitectura de aplicaciones, como la arquitectura basada en capas, la arquitectura MVC (Modelo-Vista-Controlador), la arquitectura basada en microservicios y la arquitectura basada en eventos. Cada tipo de

arquitectura tiene sus ventajas y desventajas, y la elección depende del tipo de aplicación que se desarrolle y sus requisitos [7].

La arquitectura basada en capas es una de las arquitecturas más comunes, donde una aplicación se divide en capas que realizan tareas específicas. La capa de presentación maneja la interfaz de usuario, la capa de aplicación maneja las solicitudes y la capa de datos maneja el almacenamiento y la recuperación de datos. Una arquitectura basada en capas es fácil de entender y mantener, pero no es tan escalable como otras arquitecturas [8].

La arquitectura MVC es una arquitectura que separa la lógica de la aplicación en tres componentes: modelo, vista y controlador. El modelo representa los datos y la lógica de la aplicación, la vista muestra los datos al usuario y el controlador procesa la entrada del usuario y actualiza el modelo y la vista. La arquitectura MVC es popular en el desarrollo de aplicaciones web y es conocida por su escalabilidad y facilidad de mantenimiento [8].

¿Como crear una arquitectura de una aplicación?

El proceso de diseñar una estructura es un acto de creación. Uno de los desafíos para los arquitectos es equilibrar la creatividad con el pragmatismo utilizando los métodos disponibles de modelos, marcos y formas modelo. La arquitectura puede referirse a un producto (arquitectura de edificios) o un enfoque o estilo (arquitectura de rascacielos). Además, la arquitectura debe ser reconfigurable para cumplir con los entornos dinámicos y los requisitos del cliente [9].

Hay muchas formas de crear aplicaciones y no existe una solución única para todos. Algunos factores para considerar al diseñar una arquitectura de aplicación son el tamaño y la complejidad de la aplicación, el entorno operativo (como un servidor web o un dispositivo móvil) y los requisitos de rendimiento y escalabilidad. La arquitectura de una aplicación se puede considerar como una serie de capas, cada una de las cuales es responsable de un conjunto específico de tareas. Por ejemplo, una arquitectura típica podría incluir capas para la interfaz de usuario, la lógica comercial, el acceso a datos y la comunicación con otros sistemas. Cada capa se comunica con capas adyacentes a través de interfaces bien definidas, lo que permite que cada capa sea independiente de otras capas [10].

Por otro lado, hay muchos factores a considerar al diseñar la arquitectura de una aplicación, incluyendo el tamaño y la complejidad de la aplicación, el entorno en el que se ejecutará (por ejemplo, un servidor web o un dispositivo móvil), y las necesidades de rendimiento y escalabilidad [6]. Algunos ejemplos comunes de arquitecturas de aplicaciones son la arquitectura basada en cliente-servidor, la arquitectura basada en microservicios y la arquitectura basada en contenedores [8].

Ejemplos de arquitecturas de aplicaciones.

- **Arquitectura orientada a servicios:** La arquitectura se basa en dividir la funcionalidad de una aplicación en servicios independientes que se comunican entre sí a través de una red. Esta arquitectura es común en las aplicaciones

comerciales y se puede implementar utilizando tecnologías como REST o SOAP [11].

- **Arquitectura basada en cliente-servidor:** La arquitectura cliente-servidor se puede implementar de diferentes maneras. Esto puede ser, por ejemplo, una arquitectura de dos niveles donde los clientes y servidores se comunican directamente. También puede ser una arquitectura de tres niveles, donde se agrega una capa intermedia (a menudo llamada "capa de presentación" o "capa de aplicación") entre el cliente y el servidor para permitir una mayor separación de funciones [2].
- **Arquitectura basada en microservicios:** Se basa en dividir la funcionalidad de la aplicación en microservicios independientes, que son pequeñas unidades de software independientes que se comunican entre sí a través de una red. Esta arquitectura es popular para aplicaciones escalables y se puede implementar mediante tecnologías como Docker o Kubernetes [11].
- **Arquitectura basada en contenedores:** Son una de las arquitecturas de sistemas distribuidos más populares en la actualidad. En esta arquitectura, las aplicaciones se dividen en pequeñas unidades de software denominadas contenedores que se ejecutan de forma independiente en un sistema operativo compartido. Cada contenedor contiene todo lo necesario para ejecutar una aplicación, incluidas bibliotecas, código y dependencias [12].

Beneficios de la Arquitectura de Aplicaciones.

Las arquitecturas de aplicaciones distribuidas tienen varias ventajas importantes sobre otras arquitecturas de software. Las siguientes son algunos de los beneficios comunes de la arquitectura de aplicaciones distribuidas [2]:

- **Escalabilidad:** Escalan más fácilmente a medida que crecen las demandas de los usuarios porque las diferentes partes del sistema se pueden implementar y escalar de forma independiente [2].
- **Resiliencia:** Son más tolerantes a fallas porque las fallas en un componente del sistema no necesariamente afectan a todo el sistema [13].
- **Flexibilidad:** Permite a los desarrolladores crear aplicaciones flexibles y adaptables que se pueden adaptar fácilmente a medida que cambian las necesidades de la organización [1].
- **Modularidad:** Promueven la modularidad del software, lo que significa que los desarrolladores pueden crear componentes de software pequeños y reutilizables que pueden integrarse en diferentes sistemas [14].

2.2 Arquitecturas Web

Según García, una arquitectura web es un enfoque utilizado en la construcción de sistemas informáticos, donde se utilizan componentes que se comunican entre sí para brindar servicios específicos a una organización o empresa. Esta arquitectura sigue una serie de reglas y procesos establecidos que permiten una comunicación fluida entre los componentes, lo que garantiza una mayor eficiencia y un mejor rendimiento del sistema en general [15].

Mientras otro autor nos brinda un concepto más simplificado, una arquitectura web es una arquitectura de aplicación comúnmente utilizada para construir aplicaciones web [16].

Componentes principales de las aplicaciones web.

Las aplicaciones web en una arquitectura distribuida normalmente constan de diferentes componentes que realizan diferentes funciones [8].

- **Servidor web:** Software que se ejecuta en el servidor y que se encarga de recibir solicitudes HTTP de los clientes y responder con la información adecuada [17]. Ejemplos de servidores podrían ser los siguientes: Apache HTTP Server Project[18], nginx [19] y Microsoft ISS [20].
- **Base de datos:** Sistema de almacenamiento que permite almacenar, recuperar y manipular datos de la aplicación [15]. Se tienen algunos ejemplos como: Firebase, MySQL, Postgresql [21] y MongoDB [22].
- **API:** Conjunto de interfaces que permiten que los componentes de la aplicación se comuniquen entre sí y con otros sistemas [23].
- **Front-end:** Parte de la aplicación que interactúa directamente con el usuario, a través de la interfaz de usuario [24].
- **Back-end:** Parte de la aplicación que se ejecuta en el servidor, procesando las solicitudes recibidas y proporcionando los datos necesarios [24]. Piense en la siguiente analogía, con un restaurante como escenario, el front-end serían los clientes que revisan la carta de menú, entregada a través de los meseros (API's) y los cocineros serían el back-end.
- **Middleware:** Software que se encarga de intermediar entre distintos componentes de la aplicación, proporcionando una capa de abstracción y simplificando la comunicación [23]. CICS® y WebSphere® MQ de IBM son ejemplos de middleware [25].
- **Balanceador de carga:** Componente que se encarga de distribuir la carga de trabajo entre distintos servidores, garantizando que la aplicación pueda manejar grandes cantidades de tráfico [24]. NGINX Plus y NGINX Open Source son ejemplos de balanceadores de carga [26].

Tipos de Arquitecturas Web.

Existen varios tipos de arquitecturas web, entre ellas:

Arquitectura basada en Microservicios.

En base a lo expuesto por Microsoft [27] se dice que una arquitectura de microservicios consta de varios pequeños servicios independientes que implementan una función comercial separada en un contexto limitado. Un contexto acotado es una división natural del negocio que proporciona un límite claro donde existe el modelo de dominio. Ejemplos de plataformas que utilizan microservicios son: Netflix, Ebay y Amazon [28].

Ventajas de la arquitectura de microservicios.

Algunas ventajas de la arquitectura de microservicios en arquitecturas distribuidas son las siguiente [29]:

- Escalabilidad.
- Mantenibilidad.
- Flexibilidad.
- Tolerancia a fallos.
- Mejora en la velocidad de desarrollo.

Desventajas de la arquitectura de microservicios.

Algunas desventajas de la arquitectura de microservicios en arquitecturas distribuidas son las siguiente [10]:

- Alto consumo de memoria.
- Inversión de tiempo inicial.
- Complejidad en la gestión.
- Perfil de desarrollador.

Arquitectura Single-Page Application (SPA).

Según los autores Mikowski y Powell [30], una aplicación de una sola página (SPA) es una aplicación web que carga una sola página HTML y actualiza dinámicamente esa página a medida que el usuario interactúa con la aplicación. Esto significa que, en lugar de redirigir a otras páginas HTML, el contenido se cambia mediante una solicitud AJAX al servidor y se actualiza el DOM de la página existente. El código del lado del cliente SPA (front-end) maneja la mayor parte de la lógica de la aplicación, proporcionando una mayor interactividad y capacidad de respuesta que las aplicaciones web tradicionales de varias páginas. Además, los SPA también pueden mejorar la experiencia del usuario al cargar solo los recursos requeridos cuando sea necesario, lo que reduce la cantidad de solicitudes HTTP y el tiempo total de carga de la página.

Se pueden crear este tipo de páginas con un marco de diseño como Angular [31].

Utilidad de SPA.

Una single page application SPA se puede utilizar para desarrollar aplicaciones web altamente dinámicas e interactivas. Utilizando tecnologías de próxima generación como JavaScript, CSS y HTML5, pueden cargarse y ejecutarse más rápido que las aplicaciones web tradicionales. Además, al usar un solo archivo HTML y cambiar dinámicamente su contenido, SPA reduce la cantidad de datos que deben enviarse entre el servidor y el cliente, aumentando la velocidad de carga y reduciendo el consumo de ancho de banda [32].

Ventajas de SPA.

Los beneficios o ventajas de una SPA incluyen una experiencia de usuario más fluida porque toda la aplicación se carga en una sola página, por lo que las transiciones entre páginas son más rápidas y fluidas. También hay más flexibilidad para crear diseños personalizados y aplicar animaciones personalizadas, ya que no está limitado a los

elementos de diseño predeterminados del navegador. Además, SPA funciona tanto en línea como fuera de línea, ya que los archivos se almacenan en el dispositivo del usuario [30].

Desventajas de SPA .

Las desventajas de una SPA incluyen una mayor complejidad de implementación y mantenimiento, ya que todo el código está en una sola página. El proceso de carga inicial también puede ser más lento porque todo el código de la aplicación debe descargarse a la vez. Los SPA también son menos amigables para los motores de búsqueda porque toda la información está en una página, lo que dificulta indexar el contenido para encontrarlo en los resultados de búsqueda [33].

Arquitectura basada en eventos.

La arquitectura orientada a eventos es ampliamente utilizada en aplicaciones modernas basadas en microservicios, y se basa en eventos para facilitar la comunicación entre servicios desacoplados, un evento es una notificación de cambio de estado. Esta arquitectura se compone de tres componentes principales: el productor de eventos, el enrutador de eventos y el consumidor de eventos. El productor publica un evento en el enrutador, que luego filtra y envía los eventos a los consumidores apropiados. La comunicación entre los servicios del productor y los del consumidor se realiza de manera independiente [34].

Ventajas.

Según Amazon Web Services (AWS) [34], la arquitectura basada en eventos tiene los siguientes beneficios:

- **Escalado y errores por separado.** Al desacoplar los servicios, estos solo conocen el enrutador de eventos, no los demás. Esto significa que sus servicios son interoperables, pero si un servicio tiene un error, el resto seguirá funcionando [34].
- **Auditar con facilidad.** Un enrutador de eventos actúa como una ubicación centralizada para auditar su aplicación y definir políticas.
- **Desarrollar con agilidad.** El enrutador también elimina la necesidad de una fuerte coordinación entre los servicios productores y consumidores [34].
- **Reducción de costos.** Las arquitecturas basadas en eventos son de tipo push, por lo que todo ocurre bajo demanda cuando el evento se presenta en el enrutador.

Arquitectura basada en servicios.

De acuerdo con las ideas planteadas por el autor Valencia [35], una arquitectura basada en servicios permite que ciertos servicios sean dinámicamente descritos, publicados, descubiertos e invocados en un ambiente de computación distribuida.

Un ejemplo de una plataforma basada en servicios es Amazon Web Services, que además ofrece una amplia variedad de servicios y herramientas para la construcción y gestión de aplicaciones y servicios en línea [36].

Ventajas de la arquitectura de servicios.

Según Rosado y Jaimes [37], algunas ventajas de la arquitectura de servicios son.

- Promueve la integración de tecnologías diferentes.
- Alinea y acerca las áreas de tecnología y negocio.
- Ayuda a mejorar la agilidad y flexibilidad de las organizaciones.

Arquitectura de Aplicaciones Web Progresivas (PWA, por sus siglas en inglés) .

En base a Caihuara [38], las PWA son un enfoque para desarrollar aplicaciones web que brindan una experiencia de usuario similar a la de una aplicación nativa, incrementando su funcionalidad, conforme las capacidades del dispositivo en el que se ejecuta, de ahí la palabra progresiva, web por qué se hace referencia a su desarrollo basado en tecnologías web. Un ejemplo de aplicaciones con esta arquitectura pueden ser Twitter Lite y Pinterest.

Arquitectura sin servidor.

Según Castro [39], las arquitecturas de software sin servidor combinan servicios administrados para crear aplicaciones que no requieren de la administración de infraestructura, hacen uso de servicios de cómputo para ejecutar código en respuesta a eventos sin aprovisionar ni administrar servidores en base a la tarifa de consumo, definida por el tiempo de ejecución.

Un ejemplo de este tipo de plataformas es Google Cloud, una herramienta de Google para desarrollar aplicaciones controladas por eventos utilizando una solución de computación sin servidores [40].

Arquitectura de contenedores.

En el modelo de contenedor, una instancia de imagen de contenedor representa un solo proceso. Al definir una imagen de contenedor como un límite de proceso, puede crear primitivas que se pueden usar para escalar o procesar por lotes. Cuando diseñe una imagen de contenedor, verá una definición de ENTRYPOINT en el Dockerfile. Esta definición define el proceso cuya vida útil controla la vida útil del contenedor. cuando el proceso completa, el ciclo de vida del contenedor finaliza. Los contenedores pueden representar procesos de ejecución prolongada, como servidores web, pero también pueden representar procesos de corta duración, como trabajos por lotes, que anteriormente podrían haberse implementado como Azure WebJobs [41].

Un ejemplo de arquitectura de contenedores podría ser Docker. Esta arquitectura de contenedores permite empaquetar una aplicación con todas sus dependencias en un contenedor, lo que facilita su despliegue en distintos entornos de producción. Los contenedores Docker pueden ejecutarse en diferentes sistemas operativos y en distintos tipos de nube, lo que los hace muy versátiles y adaptables a diferentes necesidades empresariales. Además, Docker proporciona una gran cantidad de herramientas y características para la gestión y el monitoreo de los contenedores, lo que lo convierte en una solución atractiva para muchas empresas que buscan una arquitectura de contenedores escalable y fácil de usar [42].

Arquitectura de contenedores.

La arquitectura de aplicación monolítica es un enfoque tradicional de diseño de aplicaciones donde todos los componentes de una aplicación se desarrollan, se prueban y se implementan como una sola unidad. En esta arquitectura, todas las funciones y características de la aplicación se ejecutan en un solo proceso, y generalmente en una sola máquina. Esto significa que cualquier cambio o actualización en la aplicación debe realizarse en toda la aplicación, lo que puede ser complejo y requiere tiempo [43].

2.3 Arquitecturas Distribuidas

Las arquitecturas distribuidas son un conjunto de patrones de diseño y estructuras para el desarrollo de sistemas de software que se ejecutan en entornos distribuidos, en los que diferentes componentes se comunican a través de redes y se ubican en distintas máquinas físicas o virtuales. Estas arquitecturas son esenciales para la construcción de aplicaciones escalables, resilientes y eficientes en entornos de nube, IoT (Internet de las cosas), Big Data y aplicaciones móviles, entre otros [10].

Las arquitecturas distribuidas permiten una mayor flexibilidad, permitiendo que las aplicaciones se adapten a diferentes entornos, se ejecuten en diferentes plataformas y se integren con sistemas y servicios externos. Además, ofrecen la posibilidad de mejorar la disponibilidad y el rendimiento de las aplicaciones, ya que permiten distribuir la carga de trabajo entre diferentes servidores y recursos de hardware [44].

Una de las arquitecturas distribuidas más populares es la arquitectura basada en microservicios, que se basa en la creación de servicios autónomos e independientes que se comunican entre sí a través de APIs. Esta arquitectura se enfoca en el modularidad, lo que permite el desarrollo, despliegue y mantenimiento de aplicaciones de manera más eficiente. Otra arquitectura importante es la basada en contenedores, que permite la ejecución de aplicaciones en entornos aislados y escalables [45].

Instancias de arquitecturas.

Se refiere a una implementación específica de una arquitectura distribuida en un entorno determinado. Por lo general, esto implica configurar y ejecutar componentes arquitectónicos en un conjunto de servidores o máquinas virtuales que funcionan juntas para proporcionar los servicios necesarios [46].

Las arquitecturas de instancias distribuidas son cada vez más comunes en el desarrollo de aplicaciones modernas, donde la escalabilidad y la disponibilidad son requisitos clave. Por ejemplo, una empresa que brinda un servicio en línea puede usar un caso de arquitectura distribuida para garantizar que el servicio esté siempre disponible para los clientes, incluso durante los períodos pico. Además, la arquitectura de instancias distribuidas se puede utilizar para aumentar la velocidad y la eficiencia de las aplicaciones mediante la distribución de la carga de trabajo entre los servidores [47].

Arquitecturas Cliente-Servidor.

La arquitectura cliente-servidor es un patrón de diseño de software que divide una aplicación en dos partes: un cliente y un servidor. En este modelo, el cliente es responsable de interactuar con los usuarios finales y brindar información, mientras que el servidor es responsable del procesamiento de datos y la lógica comercial. Los clientes y servidores se comunican a través de la red para realizar sus respectivas funciones [2].

La arquitectura cliente-servidor se utiliza en muchas aplicaciones comunes, desde aplicaciones empresariales hasta aplicaciones web. En este modelo, el servidor es el principal componente arquitectónico responsable de proporcionar servicios y recursos a los clientes. Por otro lado, los clientes son usuarios que acceden a los servicios del servidor a través de una interfaz de usuario [2].

Esta arquitectura tiene varias ventajas, como la capacidad de escalar agregando más servidores según sea necesario, el control y la gestión de datos centralizados y la separación de la lógica comercial de los clientes, lo que permite una mayor flexibilidad en el desarrollo y mantenimiento de aplicaciones [48].

Implementación de aplicaciones en tres capas.

Una implementación de aplicación de tres niveles es una arquitectura de aplicación de software que divide la funcionalidad de una aplicación en tres capas lógicas: presentación, lógica comercial y almacenamiento de datos. Esta separación permite una mayor modularidad y flexibilidad en el desarrollo de aplicaciones, así como una mejor escalabilidad y mantenibilidad del sistema [49].

La capa de presentación es responsable de la interfaz de usuario y la interacción del usuario con la aplicación. La capa de lógica empresarial maneja la funcionalidad central de la aplicación y realiza la manipulación y el procesamiento de datos. La capa de almacenamiento de datos es responsable de almacenar y recuperar datos del sistema [50].

Además de los diferentes tipos de arquitecturas distribuidas, la computación paralela es otro aspecto importante en los sistemas de software modernos. Esta arquitectura se ha vuelto muy popular en el desarrollo de aplicaciones web porque permite una clara separación de responsabilidades entre el desarrollo y el mantenimiento de la aplicación. También facilita la implementación de servicios web y aplicaciones móviles que consumen datos de aplicaciones [50].

Tipos de arquitecturas distribuidas.

Existen dos tipos de arquitecturas distribuidas que se han desarrollado y popularizado en la última década [51], las cuales son las siguientes:

- **Computación Grid:** Es una evolución de las arquitecturas de memoria distribuida, equivalente a un cluster distribuido geográficamente, heterogéneo y al cargo de dominios administrativos diferentes [51].
- **Computación Cloud:** Es un nuevo modelo de prestación de servicios de computación o de negocio a través de la virtualización de recursos, que obliga a que por debajo se implemente un sistema distribuido, y que permite al usuario acceder a un catálogo

de servicios estandarizados y trabajar en un entorno que se adapte dinámicamente a sus necesidades técnicas [51].

Además de los diferentes tipos de arquitecturas distribuidas, la computación paralela es otro aspecto importante en los sistemas de software modernos, en la siguiente sección se explicará ampliamente conceptos sobre Computación paralela.

3 Computación paralela

La computación paralela tiene como objetivo mejorar el rendimiento y la eficiencia de los sistemas informáticos al reducir el tiempo necesario para completar una tarea. Al utilizar múltiples procesadores para realizar una tarea, se puede reducir el tiempo total de procesamiento al dividir la carga de trabajo entre varios procesadores [52].

Según Ridgway Scott et al en su libro [53], menciona que la computación paralela es el proceso de dividir un problema más grande en partes más pequeñas, independientes y, a menudo, similares que pueden ejecutarse simultáneamente por múltiples procesadores que se comunican usando memoria compartida, y cuyos resultados se completan como parte del algoritmo general combinado.

Sin embargo, se utiliza en una variedad de aplicaciones, como la simulación de sistemas complejos, la investigación científica, el procesamiento de imágenes y video, el análisis de grandes conjuntos de datos y la inteligencia artificial [53].

Existen diferentes modelos de computación paralela, que varían en la forma en que se dividen las tareas y se coordinan los procesadores. Algunos de los modelos más comunes son el modelo de memoria compartida, el modelo de memoria distribuida y el modelo de procesamiento en red [54].

3.1 Modelos de computación paralela

Existen varios modelos de computación paralela que se utilizan para dividir tareas en partes más pequeñas y coordinar múltiples procesadores [55]. Estos modelos difieren en cómo se dividen las tareas y cómo se coordinan los procesadores. A continuación, se presentan algunos de los modelos más comunes de computación paralela:

- **Modelo de memoria compartida:** En este modelo, los procesadores comparten un espacio de memoria común al que todos los procesadores pueden acceder. Cada procesador tiene su propia memoria caché, pero todas las cachés están sincronizadas con la memoria principal. Los procesadores pueden leer y escribir datos en la memoria compartida, lo que permite la comunicación y la coordinación entre los procesadores [54].
- **Modelo de memoria distribuida:** En este modelo, cada procesador tiene su propia memoria y los procesadores se comunican entre sí enviando y recibiendo mensajes. Los procesadores no comparten una memoria común y, por lo tanto, la comunicación se realiza a través de la red de interconexión [55].
- **Modelo de procesamiento en red:** En este modelo, los procesadores están conectados en una red de interconexión y se comunican enviando y recibiendo

mensajes a través de la red. Cada procesador tiene su propia memoria y no hay una memoria compartida [54].

- **Modelo de procesamiento en flujo de datos:** Este modelo se utiliza para procesar flujos de datos continuos en lugar de tareas discretas. Los datos se dividen en bloques que se procesan en paralelo en múltiples procesadores. Cada procesador procesa un flujo de datos continuo y produce un flujo de datos continuo de salida [55].

3.2 Ventajas de computación paralela

La computación paralela tiene varias ventajas significativas, incluyendo:

- **Mejora del rendimiento:** al dividir una tarea en partes más pequeñas que se ejecutan en múltiples procesadores al mismo tiempo, se puede reducir significativamente el tiempo necesario para completar la tarea en comparación con la ejecución en un solo procesador [4].
- **Mayor capacidad de procesamiento:** la computación paralela permite procesar grandes conjuntos de datos de manera eficiente y rápida, lo que puede ser especialmente útil para aplicaciones que requieren grandes cantidades de datos [4].
- **Mayor capacidad de resolución:** la computación paralela puede mejorar la capacidad de resolución de problemas complejos, como la simulación de sistemas complejos o la investigación científica [56].
- **Mayor eficiencia energética:** en muchos casos, la computación paralela puede reducir el consumo de energía en comparación con la ejecución en un solo procesador, lo que puede ser beneficioso para la eficiencia energética de los sistemas informáticos [56].
- **Mejora de la escalabilidad:** la computación paralela puede mejorar la escalabilidad de los sistemas informáticos, lo que significa que los sistemas pueden manejar tareas más grandes y complejas a medida que aumenta la carga de trabajo [56].

3.3 Tipos de arquitecturas en la computación paralela

A continuación, se presentan las arquitecturas existentes en la computación paralela:

Arquitecturas según instrucciones y datos.

La arquitectura según instrucciones y datos (ISA, por sus siglas en inglés) es un conjunto de instrucciones que define la interfaz entre el hardware y el software de un sistema de cómputo. Esta arquitectura se refiere a una técnica de procesamiento de datos en la que los datos y las instrucciones se almacenan en el mismo espacio de memoria y se acceden de la misma manera [57]. En este enfoque, la unidad de control utiliza la dirección de la instrucción para buscar la instrucción y la unidad de datos utiliza la misma dirección para buscar los datos. Las arquitecturas según instrucciones y datos se han utilizado ampliamente en la industria de la computación debido a su capacidad para permitir la compatibilidad de software entre diferentes plataformas de

hardware, así como a su flexibilidad para soportar diferentes tipos de aplicaciones. Sin embargo, la complejidad de las ISA ha llevado a un aumento en la complejidad del hardware y en la dificultad para mejorar el rendimiento del procesador [51].

Arquitecturas según distribución de memoria.

En una arquitectura de memoria compartida, todos los procesadores tienen acceso a la misma memoria compartida, lo que les permite compartir información y trabajar juntos en tareas comunes. Este enfoque es común en sistemas multiprocesador, donde varios procesadores trabajan juntos en una tarea común [57].

Por otro lado, en una arquitectura de memoria distribuida, cada procesador tiene su propia memoria local y se comunica con otros procesadores a través de una red. En este enfoque, cada procesador es responsable de su propia memoria y las tareas se dividen en partes para que cada procesador las realice de forma independiente. Este enfoque es común en sistemas de computación en clúster, donde varias computadoras se conectan a través de una red para realizar una tarea común [58].

3.4 Paradigmas de programación paralela

La programación en paralelo puede ser utilizada para acelerar la ejecución de programas que procesan grandes cantidades de datos, realizan cálculos intensivos o que requieren una gran cantidad de interacciones con el usuario. Algunos ejemplos de aplicaciones que se benefician de la programación en paralelo son la simulación de sistemas complejos, el procesamiento de señales de audio y vídeo, la renderización de gráficos y la minería de datos [59].

Hay varios paradigmas de programación en paralelo, incluyendo programación de memoria compartida, programación de paso de mensajes, programación basada en tareas y programación de flujo de datos. Cada paradigma tiene sus propias ventajas y desventajas, y es importante seleccionar el enfoque adecuado para el problema que se está tratando de resolver [60].

Sin embargo, la programación paralela también presenta desafíos y limitaciones que es importante considerar antes de decidir utilizarla en un proyecto. En este sentido, es importante conocer las ventajas y desventajas de la programación paralela para tomar decisiones informadas y obtener los mejores resultados.

3.5 Ventajas

- **Rendimiento mejorado:** El uso de varios procesadores o núcleos de procesamiento en paralelo puede mejorar significativamente el rendimiento de la aplicación, ya que varias tareas pueden ejecutarse simultáneamente en diferentes procesadores [3].
- **Eficiencia energética:** La programación paralela también puede mejorar la eficiencia energética al permitir que los procesadores estén menos tiempo en funcionamiento, lo que puede reducir el consumo de energía y los costos de operación [61].

- **Escalabilidad:** La programación paralela permite que las aplicaciones se escalen a medida que se agregan más procesadores, lo que puede ayudar a manejar grandes cargas de trabajo [61].
- **Solución de problemas complejos:** La programación paralela puede ayudar a resolver problemas complejos de manera más rápida y eficiente, como la simulación de sistemas complejos, la resolución de ecuaciones diferenciales, y la modelización de sistemas [3].

3.6 Desventajas

- **Complejidad:** La programación paralela puede ser más compleja que la programación secuencial, ya que requiere una comprensión profunda de la arquitectura del sistema y de los diferentes paradigmas de programación paralela [61].
- **Dificultad en la depuración:** La depuración de programas paralelos puede ser más difícil que la depuración de programas secuenciales, ya que las interacciones entre los diferentes procesadores pueden ser más difíciles de entender y depurar [3].
- **Overhead de comunicación:** La comunicación entre diferentes procesadores puede ser costosa en términos de tiempo y recursos, lo que puede reducir la eficiencia de la aplicación [61].
- **Dificultad en el diseño de algoritmos paralelos:** El diseño de algoritmos paralelos puede ser más difícil que el diseño de algoritmos secuenciales, ya que los algoritmos deben ser diseñados para que puedan ser ejecutados de manera eficiente en múltiples procesadores [3].

3.7 Hadoop: MapReduce

Hadoop es un sistema de procesamiento distribuido que utiliza dos componentes principales para procesar grandes cantidades de datos en clústeres de servidores. El sistema de archivos distribuidos de Hadoop (HDFS) es utilizado para el almacenamiento de datos, mientras que el tiempo de ejecución de MapReduce es utilizado para el procesamiento de datos. MapReduce se basa en la idea de dividir los datos en pequeñas tareas (map) y luego combinarlos para producir un único resultado (reduce), lo que permite procesar grandes cantidades de datos de manera eficiente. Hadoop es ampliamente utilizado por empresas grandes y pequeñas para procesar grandes cantidades de datos, como registros del servidor, análisis de datos de redes sociales, análisis de datos publicitarios, y otros fines similares [62].

Hadoop MapReduce es un marco de software para escribir fácilmente aplicaciones que procesan grandes cantidades de datos (conjuntos de datos de varios terabytes) en paralelo en grandes clústeres (miles de nodos) de hardware básico de manera confiable y tolerante a fallas. Un trabajo de MapReduce generalmente divide el conjunto de datos de entrada en partes independientes que son procesadas por las tareas del mapa de manera completamente paralela. El marco ordena las salidas de los mapas, que luego se ingresan a las tareas de reducción. Normalmente, tanto la entrada

como la salida del trabajo se almacenan en un sistema de archivos. El marco se encarga de programar tareas, monitorearlas y volver a ejecutar las tareas fallidas [63].

MapReduce está diseñado para ser utilizado por programadores, en lugar de usuarios comerciales. Es un modelo de programación, no un lenguaje de programación. Ha ganado popularidad por su facilidad, eficiencia y capacidad para controlar "Big Data" de manera oportuna [64]. Además, MR permite escribir trabajos distribuidos y escalables con poco esfuerzo, a diferencia de la programación con MPL.[65] Los trabajos de MR se pueden iniciar en diferentes sistemas de archivos: HDFS, HBase, S3, sistema de archivos local, etc. MR tiene dos subcomponentes: Job Tracker (JT), Task Tracker (TT).

Según Rathinaraja [66], indica que algunos de los usos más comunes de MapReduce son los siguientes:

- Búsqueda, clasificación, agrupación.
- Estadísticas simples: contar, clasificar.
- Estadísticas complejas: PCA, covarianza.

Mientras que en el trabajo de Sikha [62]. Opina que se puede implementar de las siguientes maneras:

- Preprocesamiento de grandes cantidades de datos para aplicar algoritmos de aprendizaje automático.
- Clustering: k significa, jerárquico, densidad, bi-clustering.
- Procesamiento de texto, creación de índices, creación y análisis de gráficos, reconocimiento de patrones, filtrado colaborativo, análisis de sentimientos.

Proceso de MapReduce.

Hadoop MapReduce tiene como objetivo facilitar a los usuarios al proporcionar una abstracción limpia para los programadores al probar la paralelización automática de los programas y al proporcionar soporte de tolerancia a fallas administrado por el marco [67].

El proceso de MapReduce se divide en tres pasos principales:

- **Map:** En este paso, los datos se dividen en bloques más pequeños y se asignan a los nodos del clúster. Cada nodo ejecuta una tarea de map que procesa los datos asignados a él y produce un conjunto de pares clave-valor como salida. El resultado de este paso es una lista de pares clave-valor generados por todos los nodos [65].
- **Shuffle:** En este paso, los pares clave-valor se distribuyen por el clúster de forma que los pares con la misma clave se envían al mismo nodo de reducción. Este paso es necesario para asegurarse de que los datos con la misma clave sean procesados por el mismo nodo de reducción [67].
- **Reduce:** En este paso, los datos se agrupan por clave y se procesan por el nodo de reducción correspondiente. Cada nodo de reducción procesa los datos para una clave específica y produce un resultado final para esa clave. Los resultados finales de cada nodo de reducción se combinan para producir el resultado final de la tarea de MapReduce [65].

En un programa MapReduce típico, los usuarios solo tienen que implementar las funciones Map y Reduce y Hadoop se encarga de programarlas y ejecutarlas en paralelo. Hadoop volverá a ejecutar cualquier tarea fallida y también proporcionará medidas para mitigar cualquier cálculo desequilibrado [68].

4 Computación distribuida

La computación distribuida puede utilizarse para una amplia variedad de aplicaciones, como procesamiento de grandes cantidades de datos, simulaciones científicas, cómputo en la nube, aprendizaje automático, minería de datos, juegos en línea y sistemas de control de procesos, entre otros. La computación distribuida permite a los usuarios trabajar con grandes cantidades de datos y procesar tareas de manera más rápida y eficiente que en sistemas centralizados tradicionales.

Un ejemplo de computación distribuida es el proyecto SETI@home, que utiliza la capacidad de procesamiento no utilizada de las computadoras personales de los voluntarios para analizar señales de radio extraterrestres. Otro ejemplo es el uso de la computación distribuida en la investigación científica, donde se utilizan redes de computadoras para analizar grandes cantidades de datos y realizar simulaciones complejas [1].

4.1 Modelos de computación distribuida

Los modelos de computación distribuida son las metodologías utilizadas para desarrollar sistemas distribuidos que permiten que varias computadoras se comuniquen y coordinen sus actividades para lograr un objetivo común [69].

Los modelos generalmente involucran un conjunto de nodos que interactúan entre sí utilizando un conjunto de protocolos y algoritmos de comunicación [70]. Los nodos pueden ser computadoras, dispositivos móviles o cualquier otro tipo de dispositivo computacional.

Los modelos de computación distribuida más comunes incluyen cliente-servidor, peer-to-peer, computación móvil y computación grid [69].

- El modelo cliente-servidor implica uno o más clientes que envían solicitudes a uno o más servidores. El servidor procesa las solicitudes y devuelve los resultados.
- El modelo peer-to-peer involucra múltiples nodos que están conectados a otros nodos en una red, sin necesidad de un servidor.
- La computación móvil es un modelo de computación distribuida en el que los nodos están conectados entre sí mediante redes de comunicación inalámbrica.
- Grid Computing es un tipo de modelo de computación distribuida en el que varias computadoras están conectadas para formar una supercomputadora virtual.

Los modelos de computación distribuida proporcionan una forma de ver y analizar el comportamiento de los sistemas distribuidos [70]. Se pueden utilizar para estudiar el rendimiento de los sistemas distribuidos y para diseñar algoritmos y protocolos eficientes que se utilizan para implementar sistemas distribuidos.

4.2 RPC y RMI

La llamada a procedimiento remoto (RPC) y la invocación a método remoto (RMI) son dos de los mecanismos de comunicación entre procesos (IPC) más utilizados [2]. Tanto RPC como RMI se utilizan para facilitar la comunicación entre diferentes aplicaciones ubicadas en la misma computadora o en diferentes computadoras [71].

RPC.

Es un tipo de mecanismo de comunicación entre procesos (IPC) que permite que dos aplicaciones ubicadas en la misma computadora o en diferentes computadoras se comuniquen entre sí. Funciona llamando a un procedimiento, o función, ubicado en otra aplicación. El procedimiento puede estar ubicado en la misma computadora o en una computadora diferente. La aplicación que llama no tiene acceso directo al procedimiento remoto, sino que utiliza un proxy que actúa como intermediario entre las dos aplicaciones [72]. A continuación, se ejecuta el procedimiento remoto y los resultados se devuelven a la aplicación que realiza la llamada.

Tipos de RPC.

Según la documentación de Microsoft [73], existen los siguientes tipos:

- Solicitud/respuesta: Usado en métodos simples de solicitud/respuesta que toman y devuelven pequeñas cantidades de datos
- Dúplex
 - Dúplex unidireccional con sesión: Pueden crear una conexión persistente entre el cliente y el servidor. El servidor puede enviar datos de forma asincrónica al cliente hasta que se cierre la conexión.
 - Dúplex completo con sesión: Es un tipo de comunicación RPC en el que se establece una sesión bidireccional entre el cliente y el servidor, lo que permite que ambos envíen y reciban mensajes en cualquier momento.
- Unidireccional: Este tipo de RPC es aquel en el que un cliente envía una solicitud a un servidor y espera una respuesta única y final. El servidor no envía ninguna respuesta intermedia.

RMI.

RMI significa (Remote-Method-Invocation). Es un mecanismo que permite que un objeto que reside en un sistema (JVM) acceda/invoque un objeto que se ejecuta en otra JVM. RMI se utiliza para crear aplicaciones distribuidas; proporciona comunicación remota entre programas Java. Se proporciona en el paquete java.rmi [74].

4.3 Ventajas y desventajas

Ventajas.

- **Escalabilidad:** La computación distribuida permite agregar o reducir recursos de manera dinámica según sea necesario, lo que permite manejar grandes volúmenes de datos y carga de trabajo [75].
- **Disponibilidad:** Al distribuir el procesamiento de datos en múltiples sistemas, se reduce el riesgo de fallas catastróficas y se asegura la disponibilidad continua del sistema [55].
- **Eficiencia energética:** Al distribuir el procesamiento de datos en varios sistemas, se reduce la carga en cada sistema individual, lo que puede resultar en un menor consumo de energía y un mejor uso de los recursos [55].
- **Flexibilidad:** La computación distribuida permite la creación de sistemas personalizados y flexibles, que pueden adaptarse a diferentes requisitos de procesamiento de datos [75].

Desventajas.

- **Complejidad:** La implementación de sistemas de computación distribuida puede ser compleja y costosa, ya que implica la interconexión de múltiples sistemas y la configuración de software y hardware [55].
- **Coordinación:** La coordinación de múltiples sistemas puede ser difícil, especialmente cuando se trata de garantizar la coherencia y la integridad de los datos en sistemas distribuidos [75].
- **Seguridad:** La seguridad en sistemas de computación distribuida puede ser un desafío, ya que implica garantizar la confidencialidad, integridad y disponibilidad de los datos en múltiples sistemas [55].
- **Dependencia de la red:** La computación distribuida depende en gran medida de la conectividad de red, lo que puede ser un problema en caso de fallas en la red o en la comunicación entre sistemas distribuidos [75].

4.4 Kubernetes

Kubernetes es una plataforma de orquestación de contenedores de código abierto que fue desarrollada por Google. Es una herramienta que ayuda a automatizar el despliegue, la escalabilidad y la gestión de aplicaciones en contenedores [76].

Es una potente herramienta que puede hacer una aplicación sea más fácil de desarrollar, más rápida de desplegar y más fiable a la hora de desplegar [77].

Kubernetes es un sistema que permite una implementación basada en contenedores dentro de las nubes de plataforma como servicio (PaaS), centrándose específicamente en los sistemas basados en clústeres. Puede proporcionar una aplicación nativa de la nube (CNA), un sistema distribuido y escalable horizontalmente compuesto por (micro) servicios, con capacidades operativas como soporte de resiliencia y elasticidad. Desde un punto de vista arquitectónico Kubernetes presenta el concepto de pod, un grupo de uno o más contenedores (por ejemplo, Docker o cualquier sistema de contenedores compatible con OCI) con almacenamiento y red compartidos [78].

Arquitectura de Kubernetes.

Un clúster de Kubernetes principalmente tiene dos tipos de servidores (virtuales y físicos). En el trabajo de Aly Saleh [79] proporciona las diferencias y usos de estos dos tipos de servidores son los siguientes:

- **Máster:** El master hace funciones de controlador de clúster. Este host controla que nodo tendrá que contenedor, que recursos se pueden usar, etc. En un clúster puede existir varios masters para mantener alta disponibilidad.
- **Nodos:** Son maquina físicas o virtuales que pueden correr contenedores. Deben tener instalada un container runtime, es decir tener instalado Docker o Contained.

Componentes de servidor máster.

Dentro de un host de tipo master de Kubernetes, tenemos los siguientes componentes:

- **API Server:** Es capaz de atender peticiones API REST. Estas peticiones pueden ser bien mediante la herramienta kubectl que la herramienta de hacer llamadas, pero se puede hacer directamente utilizando consultas HTTP con datos en formato JSON.
- **Kube-Scheduler o Scheduler:** Es el programador de tareas del clúster. Se encarga de decidir que Nodo es válido para cada Pod (unidad básica de implementación y escalado) de acuerdo a los recursos disponibles [79].
- **Etdcd:** Es una base de clave-valor. Sirve para guardar las configuraciones y datos necesarios para el correcto funcionamiento interno del clúster de Kubernetes. De esta base de datos se saca Kubernetes el estado del clúster como versiones de contenedores ejecutándose, cambios en despliegues, etc [80].
- **Cloud-controller-manager:** Este componente se ejecuta a la vez que los otros componentes del nodo “master”, está basado en un sistema de plugin que permite que múltiples proveedores se integren con un clúster [80].

Componentes de servidor Nodo.

- **Kubelet:** Es el agente controlador de nodos, es responsable de inicializar un nodo. También es el encargado de comunicarse con el máster (enviar y recibir información). Es decir, envía estadísticas de uso, memoria, estado de container runtime en el nodo, etc. La comunicación con el máster contra el que nos comunicamos [79].
- **Kube-proxy:** Se encarga de la capa de red del nodo. Es decir, se encarga de la comunicación interna entre los contenedores, expone puertos, puede modificar reglas de IPTables, etc .

Ventajas.

- Un orquestador de contenedores nos permite gestionar los recursos de los nodos para levantar instancias de bases a la demanda de uso de los recursos por parte de la aplicación [81].

- Permite hacer despliegues más rápidos y controlados sin causar caída del servicio. Kubernetes hace bien la gestión de recursos [81].

5 Conclusiones

Las arquitecturas distribuidas y la computación paralela y distribuida son esenciales para el procesamiento de grandes volúmenes de datos de manera eficiente y escalable. La capacidad de distribuir el procesamiento de datos en múltiples sistemas interconectados y/o procesar tareas simultáneamente en múltiples procesadores ha permitido el desarrollo de sistemas informáticos avanzados que pueden manejar grandes cantidades de datos y proporcionar soluciones en tiempo real.

La adopción de las arquitecturas distribuidas y la computación paralela y distribuida está en constante aumento en una variedad de campos, incluyendo el análisis de datos, la inteligencia artificial, la simulación de sistemas complejos, la optimización de procesos industriales y muchos otros. A medida que la cantidad de datos y la complejidad de los sistemas informáticos siguen aumentando, estas técnicas se volverán cada vez más importantes para el desarrollo de soluciones avanzadas y el aprovechamiento de todo el potencial de la tecnología. Por lo tanto, es necesario estudiar estas técnicas para aprovechar los beneficios que brindan.

6 Referencias

- [1] O. A. Khashan and N. M. Khafajah, "Efficient hybrid centralized and blockchain-based authentication architecture for heterogeneous IoT systems," *Journal of King Saud University - Computer and Information Sciences*, vol. 35, no. 2, pp. 726–739, 2023, doi: <https://doi.org/10.1016/j.jksuci.2023.01.011>.
- [2] A. S. Tanenbaum, *Distributed Systems: Principles and Paradigms*. 2006.
- [3] M. J. Quinn, B. Burr, R. D. Williams, II, Milan, Montreal, New Delhi, Santiaao, Seoul, Sinaaooore, Sydnev, Taioei, Toronto, and G. Quinn, *Parallel Programming in C with MPI and OpenMP*.
- [4] G. C. Fox, R. D. Williams, and P. C. Messina, *Parallel Computing Works!*
- [5] C. Richardson, *Microservices Patterns*.
- [6] E. Guanabara, K. Ltda, E. Guanabara, and K. Ltda, *Distribute systems Concepts and desing*.
- [7] E. Guanabara, K. Ltda, E. Guanabara, and K. Ltda, *Patterns of enterprise application Architecture*.
- [8] L. Bass, P. Clements, R. Kazman, and an O. M. Company. Safari, "Software Architecture in Practice, 4th Edition," p. 464, 2021.
- [9] D. L. Bass, D. P. Clements, and D. R. Kazman, *Software Architecture in Practice 1th Edition*. 2003.
- [10] E. Guanabara, K. Ltda, E. Guanabara, and K. Ltda, "Building Microservices".

- [11] S. L. Only, “SPA: Principles of Service Design,” *Managing*, 2001.
- [12] T. Point, “Microservice Architecture,” p. 2, 2015.
- [13] K. Domdouzis, S. Andrews, and B. Akhgar, “Application of a new service-oriented architecture (SOA) paradigm on the design of a crisis management distributed system,” *International Journal of Distributed Systems and Technologies*, vol. 7, no. 2, pp. 1–17, 2016, doi: 10.4018/IJDST.2016040101.
- [14] T. Zheng, G. Chen, X. Wang, C. Chen, X. Wang, and S. Luo, “Real-time intelligent big data processing: technology, platform, and applications,” *Science China Information Sciences*, vol. 62, no. 8, pp. 1–12, 2019, doi: 10.1007/s11432-018-9834-8.
- [15] A. B. G. Mariscal, “UF2405 - Modelo de programación web y bases de datos,” *Editorial Elearning*, 2015. https://books.google.es/books?hl=es&lr=&id=Q11WDwAAQBAJ&oi=fnd&pg=PA9&dq=%22arquitectura+Web+es%22&ots=vSPvBNsOKt&sig=VaHICv4_eiOUCvS1ActL6Q7KG9g#v=onepage&q=conjunto de componentes&f=false
- [16] R. T. Fielding and R. N. Taylor, “Principled Design of the Modern Web Architecture,” *ACM Trans Internet Technol*, vol. 2, no. 2, pp. 115–150, 2002, doi: 10.1145/514183.514185.
- [17] A. Mathematics, “Web Application Architecture,” pp. 1–23, 2016.
- [18] Apache, “The Number One HTTP Server On The Internet,” 2018.
- [19] NGINX, “NGINX,” 2019. <https://nginx.org/en/>
- [20] Microsoft, “ISS version 1709,” 2022. <https://learn.microsoft.com/en-us/iis/get-started/whats-new-in-iis-10-version-1709/new-features-introduced-in-iis-10-1709>
- [21] PostgreSQL, “PostgreSQL Manual,” 2023. <https://www.postgresql.org/docs/>
- [22] MongoDB, “What is MongoDB,” 2023. <https://www.mongodb.com/docs/manual/>
- [23] M. Deinum, K. Serneels, C. Yates, S. Ladd, and C. Vanfleteren, “Web Application Architecture Guide,” *Pro Spring MVC: With Web Flow*, pp. 51–64, 2012, doi: 10.1007/978-1-4302-4156-0_3.
- [24] R. Baxter, *Software engineering a Practitioner’s Approach*. 2006. doi: 10.1049/ic:20040411.
- [25] IBM, “middleware,” 2021. <https://www.ibm.com/docs/en/tivoli-monitoring/6.3.0?topic=m-middleware>
- [26] NGINX, “What Is Load Balancing?,” 2022. <https://www.nginx.com/resources/glossary/load-balancing/>
- [27] Microsoft, “Estilo de arquitectura de microservicios,” 2022. <https://learn.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices>
- [28] OpenWebinars, “Qué son Microservicios y ejemplos reales de uso,” 2016. <https://openwebinars.net/blog/microservicios-que-son/>
- [29] M. Mazzara and B. Meyer, “Microservices: yesterday, today, and tomorrow,” *Present and Ulterior Software Engineering*, pp. 1–225, 2017, doi: 10.1007/978-3-319-67425-4.

- [30] M. S. Mikowski, J. C. Powell, and G. D. Benson, *Single Page Web Applications*.
- [31] Angular, “Introduction to the Angular docs,” 2023. <https://angular.io/docs>
- [32] E. Molin, “Comparison of Single-Page Application Frameworks: A method of how to compare Single-Page Application frameworks written in JavaScript.,” *Degree Project Computer Science and Engineering*, 2016.
- [33] S. P. Applications, “Introduction to Single Page Applications Objectives What is a Single Page Application (SPA)? Advantages of SPA Downsides of SPA How to build SPA ’ s using Angular 1 . 1 What is a Single Page Application (SPA),” pp. 1–11.
- [34] AWS, “¿Qué es la arquitectura basada en eventos?,” AWS. <https://aws.amazon.com/es/event-driven-architecture/>
- [35] C. A. Morales -Machuca, “Estado del Arte: Servicios Web,” *Camoralesmagooglepagescom*, 2010.
- [36] Amazon, “AWS,” 2022. <https://aws.amazon.com/es/what-is-aws/>
- [37] A. Rosado and J. Jaimes, “Review of the Incorporation of Service-Oriented Architecture in Organizations,” *Revista Colombiana de Tecnologías de Avanzada*, vol. 1, no. 31, 2018.
- [38] F. D. Caihuara Sossa, “Aplicaciones web progresivas,” *Mozilla*, vol. 5, no. 2015, pp. 61–67, 2019.
- [39] D. D. E. Ingeniería and D. E. S. Y. Computación, “Arquitecturas de software sin servidor : Un caso de estudio para el despliegue en Amazon Web Services , Microsoft Azure e IBM Bluemix Contenido”.
- [40] Google, “Cloud Functions y Firebase,” 2022. <https://firebase.google.com/docs/functions/functions-and-firebase?hl=es-419>
- [41] “NET-Microservices-Architecture-for-Containerized-NET-Applications”.
- [42] mairaw Jamesmontemagno, erjain, john-par, “Introducción a Containers y Docker,” *Microsoft*, 2023. <https://learn.microsoft.com/es-es/dotnet/architecture/microservices/container-docker-introduction/>
- [43] N. mairaw, john-par, gewarren, Youssef1313, sughosneo, DCtheGeek, erjain, “Aplicaciones monolíticas,” *Microsoft*, 2023. <https://learn.microsoft.com/es-es/dotnet/architecture/containerized-lifecycle/design-develop-containerized-apps/monolithic-applications>
- [44] C. Pautasso, O. Zimmermann, and F. Leymann, “RESTful web services vs. ‘Big’ web services: Making the right architectural decision,” *Proceeding of the 17th International Conference on World Wide Web 2008, WWW’08*, no. April, pp. 805–814, 2008, doi: 10.1145/1367497.1367606.
- [45] P. Di Francesco, P. Lago, and I. Malavolta, “Architecting with microservices: A systematic mapping study,” *Journal of Systems and Software*, vol. 150, pp. 77–97, 2019, doi: 10.1016/j.jss.2019.01.001.
- [46] W. H. Kersting, “Introduction to Distribution Systems,” *Distribution System Modeling and Analysis*, pp. 17–26, 2020, doi: 10.1201/b11697-5.
- [47] R. S. S. Filho, J. Wainer, and E. R. M. Madeira, “A fully distributed architecture for large scale workflow enactment,” *Int J Coop Inf Syst*, vol. 12, no. 4, pp. 411–440, 2003, doi: 10.1142/S0218843003000802.

- [48] Geeksforgeeks, “Fundamentals of Software Architecture,” *Geeksforgeeks*, 2021.
- [49] P. By and C. Kambalyal, “N-Tier Architecture,” *Pro LINQ Object Relational Mapping with C# 2008*, pp. 311–345, 2008, doi: 10.1007/978-1-4302-0597-5_11.
- [50] J. J. Donovan and O. E. Corporation, “Three-Tier Architecture”.
- [51] N. De and A. De, “Modelado del rendimiento de códigos irregulares paralelos en sistemas distribuidos,” 2015.
- [52] A. Mathematics, *Introduction to Parallel Computing, Second*. 2016.
- [53] G. E. Karniadakis and R. M. K. Ii, “Parallel Scientific Computing,” *Immunol Lett*, vol. 17, no. 1, p. 92, 1988, doi: 10.1016/0165-2478(88)90108-3.
- [54] K. Coulouris, “Distributed Systems Concepts and Design.”
- [55] B. Schmidt, J. González-Domínguez, C. Hundt, and M. Schlarb, *Parallel Programming: Concepts and Practice*. 2017. doi: 10.1016/C2015-0-02113-X.
- [56] P. J. Denning and W. F. Tichy, “Highly parallel computation,” *Science (1979)*, vol. 250, no. 4985, pp. 1217–1222, 1990, doi: 10.1126/science.250.4985.1217.
- [57] DavidB. Skillicorn, *Models and Languages for Parallel Computation*, vol. 1, no. 0.
- [58] C. Kessler and J. Keller, “Models for Parallel Computing: Review and Perspectives,” *PARS-Mitteilugen*, no. 24, pp. 13–29, 2007.
- [59] T. Nogi, “Parallel Computation,” *Studies in Mathematics and its Applications*, vol. 18, no. C, pp. 279–318, 1986, doi: 10.1016/S0168-2024(08)70134-6.
- [60] F. Gebali, *Algorithms and parallel computing*. 2011.
- [61] S. Hariri and M. Parashar, *Tools and Environments for Parallel and Distributed Computing*. 2004.
- [62] S. Bagui and P. C. Dhar, “Positive and negative association rule mining in Hadoop’s MapReduce environment,” *J Big Data*, vol. 6, no. 1, 2019, doi: 10.1186/s40537-019-0238-8.
- [63] Apache, “MapReduce Tutorial,” 2022. https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
- [64] S. Maitrey and C. K. Jha, “MapReduce: Simplified Data Analysis of Big Data,” *Procedia Comput Sci*, vol. 57, pp. 563–571, 2015, doi: <https://doi.org/10.1016/j.procs.2015.07.392>.
- [65] K. Kalia and N. Gupta, “Analysis of hadoop MapReduce scheduling in heterogeneous environment,” *Ain Shams Engineering Journal*, vol. 12, no. 1, pp. 1101–1110, 2021, doi: <https://doi.org/10.1016/j.asej.2020.06.009>.
- [66] R. Jeyaraj, G. Pugalendhi, and A. Paul, *Big Data with Hadoop MapReduce: A Classroom Approach*. Apple Academic Press, 2020.
- [67] dkk 2018) richard oliver (dalam Zeithml., *Hadoop The definitive Guide*. 2021.
- [68] J. Venner, *Pro Hadoop*.
- [69] A. S. Tanenbaum, “Distributed Computing,” pp. 1–111, 2017.
- [70] K. Srinivasa and A. Muppalla, *Guide to High Performance Distributed Computing*. 2015. doi: 10.1007/978-3-319-13497-0.

- [71] S. J. T. Condie, “Distributed Computing, Tomorrow’s Panacea — an Introduction to Current Technology,” *BT Technology Journal*, vol. 17, no. 2, pp. 13–23, 1999, doi: 10.1023/A:1009673430037.
- [72] M. R. De Borja, “Implementing Remote Procedure Calls,” *Basques in the Philippines*, vol. 2, no. 1, pp. 1–183, 2012.
- [73] M. JamesNK, esjain, gewarren, nschonni, ShawnJackson, mairaw, “Tipos de RPC,” *Microsoft*, 2022. <https://learn.microsoft.com/es-es/dotnet/architecture/grpc-for-wcf-developers/rpc-types>
- [74] M. Praveena, *Java Programming*. Shanlax Publications, 2018.
- [75] W. P. Petersen and P. Arbenz, “Introduction to Parallel Computing,” *Computing*, 2004.
- [76] Kubernetes, “¿Qué es Kubernetes?,” *Kubernetes*, 2023. <https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/>
- [77] B. Burns, E. Villalba, D. Strebel, and L. Evenson, *Guía práctica de Kubernetes*. Marcombo, 2020.
- [78] V. Medel, R. Tolosana-Calasanz, J. Á. Bañares, U. Arronategui, and O. F. Rana, “Characterising resource management performance in Kubernetes,” *Computers & Electrical Engineering*, vol. 68, pp. 286–297, 2018, doi: <https://doi.org/10.1016/j.compeleceng.2018.03.041>.
- [79] A. Saleh and M. Karslioglu, *Kubernetes in Production Best Practices: Build and manage highly available production-ready Kubernetes clusters*. Packt Publishing, 2021.
- [80] G. G. Urtiaga, *Kubernetes desde cero: Conviértete en profesional de la forma mas fácil*. AprendeIT, 2022.
- [81] F. Rossi, V. Cardellini, F. Lo Presti, and M. Nardelli, “Geo-distributed efficient deployment of containers with Kubernetes,” *Comput Commun*, vol. 159, pp. 161–174, 2020, doi: <https://doi.org/10.1016/j.comcom.2020.04.061>.

7 Prácticas realizadas en clase

Se desarrollaron dos prácticas en clase, con los siguientes temas:

- RMI o "Remote Method Invocation" (Invocación de Método Remoto).
- Threads (Hilos)

Ambas prácticas fueron desarrolladas usando el lenguaje de programación Java en el IDE Apache NetBeans 17. Puede revisar el código de RMI a profundidad en el siguiente repositorio de GitHub: [Clic Aquí](#)

7.1 Práctica sobre RMI

Para el desarrollo de esta práctica se ha elegido el ejemplo de un Banco, en donde se podrán realizar las siguientes acciones:

- Depósitos (a la cuenta personal u otro usuario)
- Retiros
- Ver saldo actual

El proyecto se conforma de cuatro clases:

- **Bank:** Clase de tipo Interface, extiende (extends) la interfaz "Remote".

Código relevante.

```
public interface Bank extends Remote{
    void deposit(int accountNumber, double amount)
    throws RemoteException;
    void withdraw(int accountNumber, double amount)
    throws RemoteException;
    double getBalance(int accountNumber) throws
    RemoteException;
}
```

- **BankImpl:** En esta clase se implementan los métodos de la clase Bank.

Código relevante.

```
public class BankImpl extends UnicastRemoteObject
implements Bank{
    private double[] accounts;

    public BankImpl() throws RemoteException {
        accounts = new double[2];
    }
    public void deposit(int accountNumber, double
    amount) throws RemoteException {
```

```

        accounts[accountNumber - 1] += amount;
    }

    public void withdraw(int accountNumber, double
amount) throws RemoteException {
        accounts[accountNumber - 1] -= amount;
    }

    public double getBalance(int accountNumber) throws
RemoteException {
        return accounts[accountNumber - 1];
    }
}

```

- **BankServer:** Clase en donde se escriben las instrucciones necesarias para el servidor del banco.

Código relevante.

```

public class BankServer {
    public static void main(String[] args) {
        try {
            Bank bank = new BankImpl();
            Registry registry =
LocateRegistry.createRegistry(1099);
            Naming.rebind("Bank", bank);
            System.out.println("Bank server ready");
            JOptionPane.showMessageDialog(null,
"Servidor del banco conectado existosamente");
        } catch (Exception e) {
            System.err.println("Bank server exception:
" + e.getMessage());
            e.printStackTrace();
        }
    }
}

```

- **BankClient:** En esta clase se encuentran definidas las opciones que se presenatran a los clientes.

Código relevante.

Se mostrará un fragmento de código en donde está el menú del banco.

```

public class BankClient {
    public static void main(String[] args) {
        try {

```

```

        Bank bank = (Bank)
Naming.lookup("rmi://localhost/Bank");

        while (true) {
            String menu =
JOptionPane.showInputDialog("      Banco RMI      \n\n
Menú del Banco      \n"
            +"1. Depósito\n"
            +"2. Retiro\n"
            +"3. Consultar saldo\n\n"
            +"Cancelar para salir\n\n");

            int accountNumber1=0;

            switch (menu) {
                case "1":
                    accountNumber1 =
Integer.parseInt( JOptionPane.showInputDialog("Escriba
número de cuenta para depositar: "));
                    double depositAmount =
Integer.parseInt( JOptionPane.showInputDialog("Escriba
el monto a depositar: "));
                    bank.deposit(accountNumber1,
depositAmount);

JOptionPane.showMessageDialog(null, "Se ha depositado
$" + depositAmount + " al Cliente " + accountNumber1);
                    break;

```

Ejecución de la práctica.

1. Se inicia ejecutando el servidor del Banco en el puerto 1099, tal como se muestra en la Fig. 1, se ejecuta y presenta un mensaje indicando que inició su actividad.

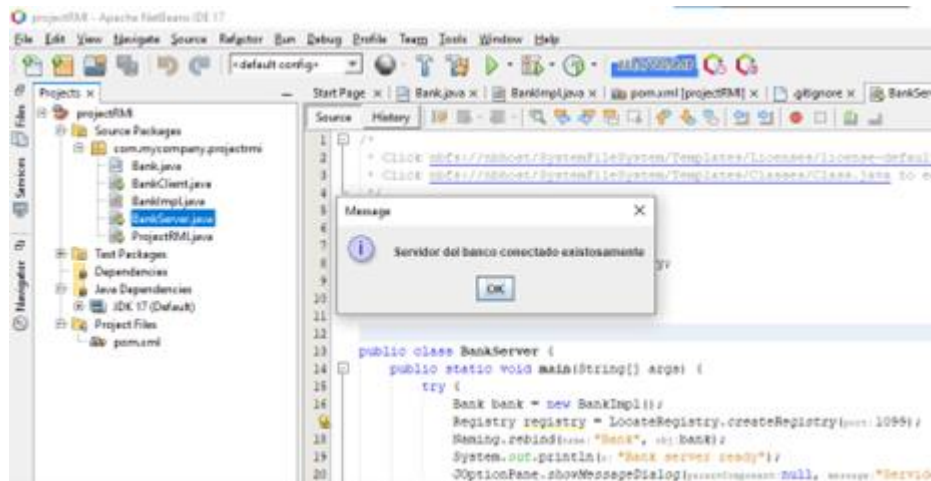


Fig. 1. Servidor del banco activo

2. Como se puede observar en la Fig. 2, en esta práctica se diseñó un menú con las acciones más importantes de un Banco.

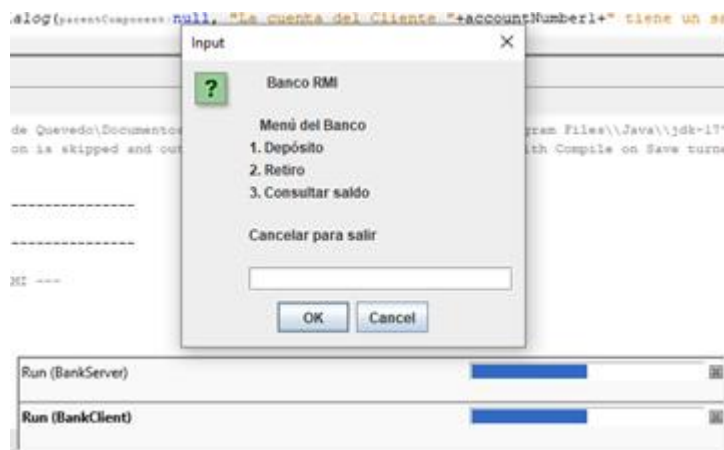


Fig. 2. Menú del banco

3. Para la ejecución del programa, existen dos clientes predefinidos (Cliente 1 y Cliente 2). A continuación, se asumirá el rol de Cliente 1 depositando a su cuenta. Como se puede observar en la Figura 3, primero debe ingresar su número de cuenta, seguido del monto a depositar (ver Fig. 4) y, por último, se le presenta al usuario un mensaje de confirmación

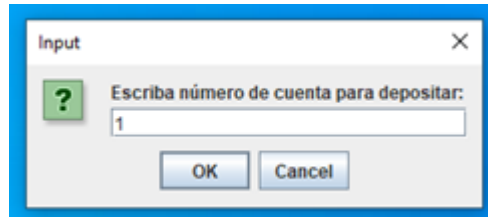
An input dialog box titled "Input" with a close button (X) in the top right corner. It contains a green question mark icon on the left. The text "Escriba número de cuenta para depositar:" is displayed above a text input field. The input field contains the number "1". Below the input field are two buttons: "OK" and "Cancel".

Fig. 3. Ingreso de número de cuenta

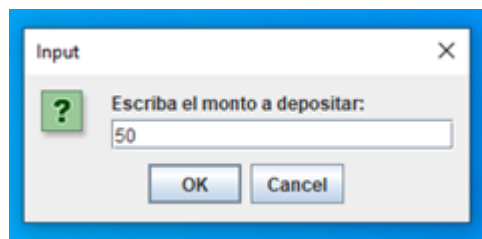
An input dialog box titled "Input" with a close button (X) in the top right corner. It contains a green question mark icon on the left. The text "Escriba el monto a depositar:" is displayed above a text input field. The input field contains the number "50". Below the input field are two buttons: "OK" and "Cancel".

Fig. 4. Ingreso de monto a depositar

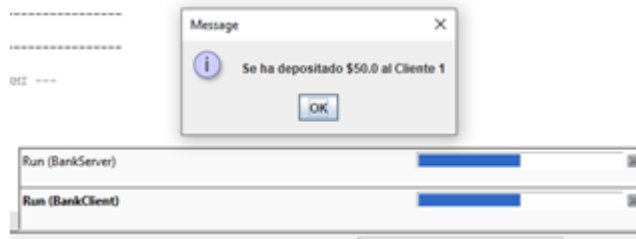


Fig. 5. Mensaje con información de la acción realizada

4. A continuación, realizando los pasos anteriores, un Cliente 2 realizará un depósito al Cliente 1 (Ver Fig. 6), proceso que se refleja en la Fig. 7.

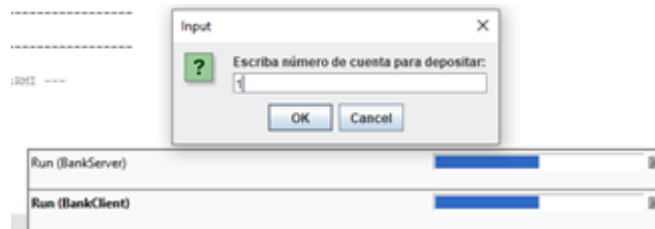


Fig. 6. Ingreso de número de cuenta

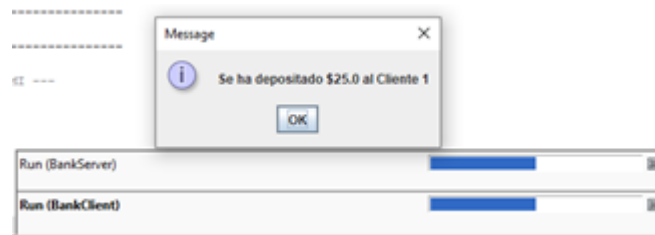


Fig. 7. Mensaje de confirmación de depósito

5. Se realizará un retiro de \$15 como se indica en la Figura 8 al Cliente 1. Luego se ingresa el número de Cuenta y el monto a retirar (Ver Fig. 9). Finalmente, se indica el monto que ha sido retirado como se muestra en la Fig. 10.

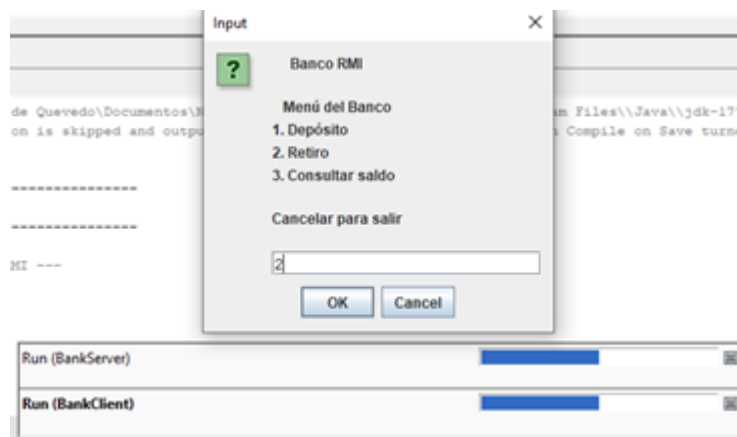


Fig. 8. Retiro de dinero

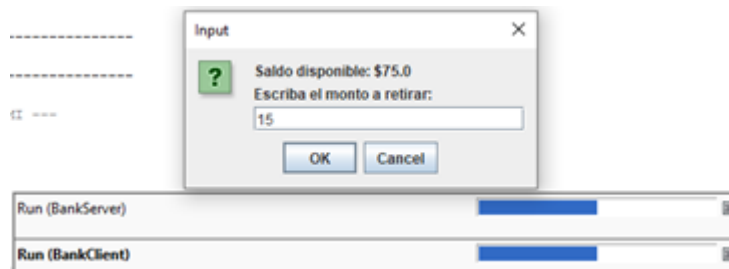


Fig. 9. Ingreso de monto a retirar

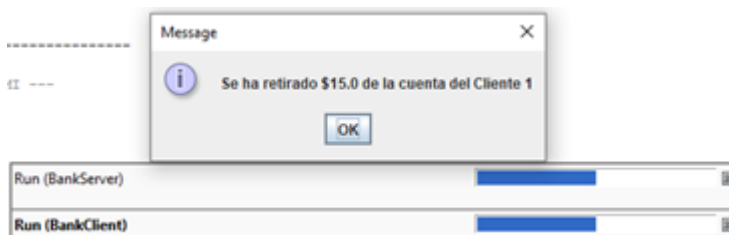


Fig. 10. Mensaje de confirmación

6. En la Figura 11, se realizará una consulta de saldo mediante el menú (Ver). El saldo para consultar será del Cliente 1 (Ver Fig. 12). Se mostrará una venta con la información solicitada como se muestra en la Figura 13.

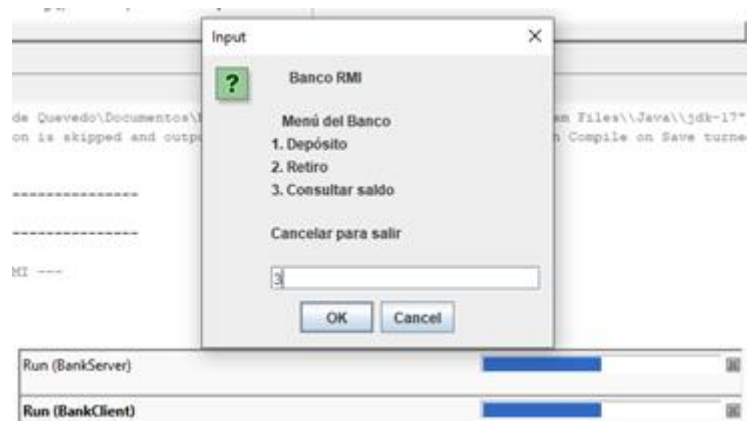


Fig. 11. Consulta de saldo

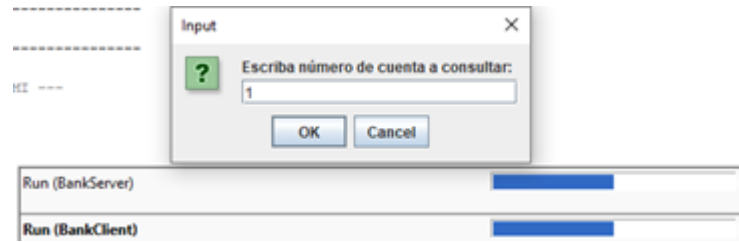


Fig. 12. Ingreso de número de cuenta

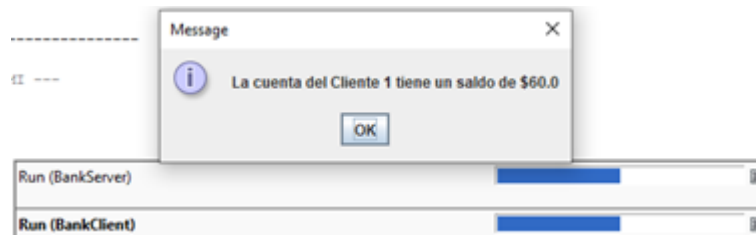


Fig. 13. Mensaje informando saldo del Cliente

7.2 Práctica sobre Threads

Para el desarrollo de esta práctica se ha elegido el ejemplo del juego del Snake, en donde se podrán realizar las siguientes acciones:

- Cambiar la dirección del Snake con las teclas direccionales
- Ver el Puntaje Actual
- Ver el tiempo jugado

El proyecto se conforme de cinco clases y el main principal:

- **GestorPuntaje:** En esta clase se encuentran definidas las operaciones para poder escribir y leer en un archivo .txt el puntaje actual.

Código relevante.

```
public void VaciarTexto() {
    try {
        File archivo = new
File("src\\Recursos/puntajes.txt");
        FileWriter escritorArchivo = new
FileWriter(archivo, false);
        escritorArchivo.write("");
        escritorArchivo.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void Escribir(String valor) {
    FileWriter escritorArchivo = null;
    try {
        File archivo = new
File("src\\Recursos/puntajes.txt");
        escritorArchivo = new
FileWriter(archivo);
        escritorArchivo.write(valor);
    } catch (IOException ex) {

Logger.getLogger(PanelSnake.class.getName()).log(Level.
SEVERE, null, ex);
    } finally {
        try {
            escritorArchivo.close();
        } catch (IOException ex) {

JOptionPane.showMessageDialog(null, ex.toString());

```

```

    }
}

public String Leer(){
    FileReader lectorArchivo = null;
    String linea="";
    String retorno="";
    try {
        File archivo = new
File("src\\Recursos/puntajes.txt");
        lectorArchivo = new FileReader(archivo);
        BufferedReader bufferLector = new
BufferedReader(lectorArchivo);

        while ((linea = bufferLector.readLine())
!= null) {
            retorno=linea;
        }

    } catch (FileNotFoundException ex) {

Logger.getLogger(PanelSnake.class.getName()).log(Level.
SEVERE, null, ex);
    } catch (IOException ex) {

Logger.getLogger(PanelSnake.class.getName()).log(Level.
SEVERE, null, ex);
    } finally {
        try {
            lectorArchivo.close();
        } catch (IOException ex) {

Logger.getLogger(PanelSnake.class.getName()).log(Level.
SEVERE, null, ex);
        }
    }

    return retorno;
}

```

- **SnakeTiempo:** En esta clase se encuentran definidas las operaciones para mostrar el tiempo, esta clase se implementa un Runnable.

Código relevante.

```

public class SnakeTiempo implements Runnable
{
    private JLabel tiempoLabel;
    boolean estado=true;
    public SnakeTiempo(JLabel tiempoLabel) {
        this.tiempoLabel = tiempoLabel;
    }

    @Override
    public void run() {
        int segundos = 0;
        int minutos = 0;
        int horas = 0;

        while (true) {
            try {
                Thread.sleep(1000); //espera un
segundo

                segundos++;

                if (segundos == 60) {
                    segundos = 0;
                    minutos++;
                }

                if (minutos == 60) {
                    minutos = 0;
                    horas++;
                }

                String tiempo = "Tiempo: "+
String.format("%02d:%02d:%02d", horas, minutos,
segundos);

                tiempoLabel.setText(tiempo);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

```

- **Caminante:** En esta clase se encuentran definidas las operaciones para mover al snake en tiempo real, esta clase se implementa un Runnable.

Código relevante.

```

public class Caminante implements Runnable{
    PanelSnake panel;
    boolean estado=true;
    public Caminante(PanelSnake panel){
        this.panel=panel;
    }

    @Override
    public void run(){
        while (estado) {
            panel.avanzar();
            panel.repaint();

            try{
                Thread.sleep(200);
            }
            catch (InterruptedException ex){

                Logger.getLogger(Caminante.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
}

```

- **PanelFondo:** Esta clase se encarga de dibujar el fondo donde se encontrara el snake, esta clase se extiende de JPanel.

```

public class PanelFondo extends JPanel
{
    Color colorfondo=Color.gray;
    int tammax,tam,can,res;

    public PanelFondo(int tanmax,int can)
    {
        this.tammax=tanmax;
        this.can=can;
        this.tam=tammax/can;
    }
}

```

```

        this.res=tammax%can;
    }

    @Override
    public void paint(Graphics pintor)
    {
        super.paint(pintor);
        pintor.setColor(colorfondo);
        for (int i = 0; i < can; i++) {
            for (int j = 0; j < can; j++) {
                pintor.fillRect(res/2+i*tam,
res/2+j*tam, tam-1, tam-1);
            }
        }
    }
}

```

- **PanelSnake:** Esta clase se encarga de dibujar al snake y la comida, además de tener operaciones que permiten cambiar la dirección del movimiento, esta clase se extiende de JPanel.

```

public PanelSnake(int tanmax, int can)
{
    this.tammax=tanmax;
    this.can=can;
    this.tam=tammax/can;
    this.res=tammax%can;
    int []a= {can/2-1, can/2-1};
    int []b= {can/2, can/2-1};
    snake.add(a);
    snake.add(b);

    generarcomida();
    camino=new Caminante(this);
    hilo=new Thread(camino);
    Gestor=new GestorPuntaje();
    Gestor.VaciarTexto();
    hilo.start();

}

@Override
public void paint(Graphics pintor)

```

```

{
    super.paint(pintor);
    pintor.setColor(colorsnake);
    //pintar snake
    for (int [] par:snake) {
        pintor.fillRect(res/2+par[0]*tam,
res/2+par[1]*tam, tam-1, tam-1);
    }
    //pintar comida
    pintor.setColor(colorcomida);
    pintor.fillRect(res/2+comida[0]*tam,
res/2+comida[1]*tam, tam-1, tam-1);
}

public void avanzar()
{

    igualarDireccion();
    int [] ultimo=snake.get(snake.size()-1);
    int agregarx=0;
    int agregary=0;

    if(estado.equals("start"))
    {
        switch(this.direccion)
        {
            case "de":
                agregarx=1;
                break;
            case "iz":
                agregarx=-1;
                break;
            case "ar":
                agregary=-1;
                break;
            case "ab":
                agregary=1;
                break;
            case "stop":
                hilo.stop();
                break;
            case "start":
                hilo.start();
                break;
        }
    }
}

```

```

        int []
nuevo={Math.floorMod(ultimo[0]+agregarx,
can),Math.floorMod(ultimo[1]+agregary, can) };
        // snake.add(nuevo);
        // snake.remove(0);
        boolean existe=false;
        for (int i = 0; i < snake.size(); i++) {

if(nuevo[0]==snake.get(i) [0]&&nuevo[1]==snake.get(i) [1]
){
                existe=true;
                break;
            }
        }
        if(existe){

        }else{

if(nuevo[0]==comida[0]&&nuevo[1]==comida[1]){
                snake.add(nuevo);
                generarcomida();
                Gestor=new GestorPuntaje();
                if(Gestor.Leer().equals("")){
                    Gestor.Escribir("10");
                }
                else{
                    int
n=Integer.parseInt(Gestor.Leer());
                    n=n+10;
                    Gestor.VaciarTexto();

Gestor.Escribir(Integer.toString(n));
                }
            }
            else{
                snake.add(nuevo);
                snake.remove(0);
            }
        }
    }

    public void generarcomida(){
        boolean existe=false;

```

```

int a= (int) (Math.random()*can);
int b= (int) (Math.random()*can);

for(int [] par:snake) {
    if(par[0]==a&&par[1]==b) {
        existe=true;
        generarcomida();
        break;
    }
}

if(!existe)
{
    this.comida[0]=a;
    this.comida[1]=b;
}

}

public void cambiardireccion(String dir)
{
    //Se comprueba si es pausa
    if(dir.equals("stop")&&estado.equals("start"))
{
        estado="stop";
    }
    else
if(dir.equals("stop")&&estado.equals("stop")) {
    estado="start";
    }
    else{
        //Evitamos repetir la tecla
        if(!(this.direccion.equals(dir)))
            this.direccionproxima=dir;
    }

}

public void igualarDireccion() {
    this.direccion=this.direccionproxima;
}

```


- **Vista:** Esta clase se encarga de invocar las clases antes mencionadas además de constar con un formulario

```

PanelSnake panel;
    Thread hilo;
    SnakeTiempo tiempo;
    String op="start";

    public Vista() {
        initComponents();
        this.setLocationRelativeTo(null);

        GestorPuntaje Gestor=new GestorPuntaje();
        panel=new PanelSnake(700, 30);
        this.add(panel);
        panel.setBounds(10,10,700,700);
        panel.setOpaque(false);

        PanelFondo Fondo=new PanelFondo(700, 30);
        this.add(Fondo);
        Fondo.setBounds(10,10,700,700);

        this.requestFocus(true);

        this.pack();
        tiempo=new SnakeTiempo(jLabel2);
        hilo =new Thread(tiempo);
        hilo.start();

        SnakePuntaje puntaje=new SnakePuntaje(jLabel3);
        Thread hil2=new Thread(puntaje);
        hil2.start();

    }

    public void MovimientoSerpiente(String estado){
        panel.cambiardireccion(estado);
        panel.avanzar();
    }

```

```

        panel.repaint();
    }

    private void formKeyPressed(java.awt.event.KeyEvent
    evt) {
        // TODO add your handling code here:
        switch (evt.getKeyCode()) {
            case KeyEvent.VK_LEFT:
                MovimientoSerpiente("iz");
                break;
            case KeyEvent.VK_RIGHT:
                MovimientoSerpiente("de");
                break;
            case KeyEvent.VK_UP:
                MovimientoSerpiente("ar");
                break;
            case KeyEvent.VK_DOWN:
                MovimientoSerpiente("ab");
                break;
            case KeyEvent.VK_SPACE:
                MovimientoSerpiente("stop");
        }
    }
}

```

Ejecución de la práctica.

Se inicia ejecutando la aplicación, esta consta de tres hilos (Ver Fig. 14):

- El primero hilo se enfoca en mover al automáticamente por el panel, y que se puede cambiar la dirección del movimiento con las teclas direccionales, además de generar nueva comida, cada vez que el snake coma
- El segundo hilo se enfoca en mostrar el tiempo total jugado en un JLabel
- Y por el ultimo el tercer hilo se enfoca en calcular y mostrar el puntaje actual.

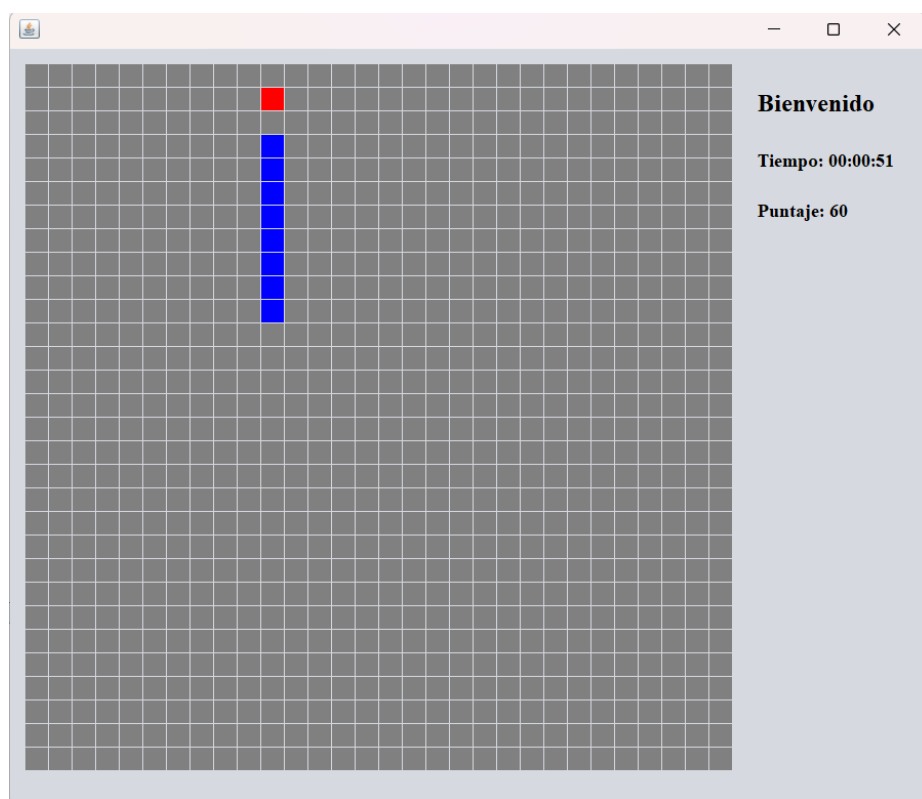


Fig. 14. Juego del snake