Estrategia implementada sistema de votaciones

Johan Daniel Aguirre Arias

Alejandro Amu Garcia

David Henao Salazar

Facultad de Ingeniería diseño y ciencias aplicadas, Universidad ICESI

Ingeniería de Software IV

Ingeniería de sistemas

2 de diciembre de 2024

Tabla de contenidos

Análisis de requerimientos funcionales haciendo uso de los métodos de dorfman.

1. Los grupos de votantes por cada ciudad se agrupan por número de cédula en mesas de votación. Las mesas están ubicadas en colegios y universidades, y para la agrupación se tiene en cuenta la dirección de residencia que el votante tendrá que haber registrado previamente en el sistema central.
2. A cualquier ciudadano se le debe permitir consultar en qué sitio debe votar, por una interfaz de usuario sencilla, ingresando el documento de identidad.
3. El sistema debe ser escalable, pero, sobre todo, altamente disponible. Se deben poder añadir más clientes en tiempo real.
4. Todas las estaciones de consulta (clientes) deben comunicarse con el sistema de consulta (servidor) usando un protocolo de comunicación ICE.
5. El servidor debe poder atender concurrentemente a todos los clientes que hagan consultas sobre la mesa de votación, con una latencia máxima de 1 segundo, aunque esta cantidad de tiempo podrá variar.
6. Para efectos de simulación y prueba, cada cliente debe implementar un pool de N consultas para ejecutar concurrentemente, donde N lo podrá ingresar el usuario final al iniciar el programa cliente.
7. Entre el cliente y el servidor debe implementarse el patrón Observer. El servidor es el observado, y los clientes son los observadores que se registran en el primero.
8. El servidor, en su rutina principal, debe ejecutar un ciclo en el que pregunte al usuario que digite un mensaje. El usuario digitara una ruta que incluye el nombre de un archivo. Este archivo tendrá una lista de documentos de identidad (uno por línea) a los que se desea consultar la mesa de votación. Si la cantidad de líneas es menor a la cantidad de consultas que un computador puede realizar sin perder consultas ni abortarse, él mismo debe hacerlas. En caso contrario, debe distribuir secciones de la lista entre los clientes registrados en él, notificándoles como sus Observadores, para que cada cliente realice las consultas de los documentos de identidad que le correspondió, sobre el servidor, en el menor tiempo posible (el tiempo estimado deseable es máximo un segundo).
9. El servidor debe calcular el número de factores primos del documento de identidad y hacer la consulta en la BD la mesa de votación. Si el número de factores primos es primo, debe retornar
10. El sistema debe ser auditable. En particular, en los clientes debe generarse un log con columnas separadas por coma. Las columnas son: la CC de consulta, y para cada CC, la respectiva mesa de votación, un indicador de si el número de factores primos es primo (1 si lo es, 0 si no lo es), y el tiempo que se demoró el sistema en responder, etc., que permita verificar la correctitud de las respuestas. En el servidor, debe dejarse la misma información por cada línea, pero antecedido del id del cliente que lo consultó, en un solo archivo de salida. En la última línea del archivo de salida, el número total de consultas realizadas y el tiempo total de ejecución.

Para realizar el análisis de los requerimientos anteriores se hizo uso de las dos fases de la metodología de dorfman más un análisis inicial para identificación de los requerimientos funcionales, no funcionales y sus respectivos RAS, para observar el proceso de manera detallada diríjase al anexo "Requerimientos del proyecto final".

Análisis de requerimientos no funcionales haciendo uso de los métodos de QAW.

Para analizar los requerimientos no funcionales detallados en el anexo "Requerimientos del proyecto final", identificamos los aspectos más críticos relacionados con los atributos de calidad. A partir de ello, desarrollamos una serie de escenarios que facilitan la comprensión y resolución de dichos aspectos utilizando el método QAW.

ESCENARIO QAW:		Durante la ejecución de las consultas, se pierde temporalmente la conexión entre el servidor y un cliente.
META DE NEGOCIO:		Evitar la pérdida de consultas y garantizar la continuidad del servicio incluso en condiciones de red inestables.
ELEMENTOS	ATRIBUTO DE CALIDAD:	Disponibilidad, Consistencia, Confiabilidad
	ESTÍMULO:	Fallo temporal en la conexión de red entre el servidor y un cliente.
	FUENTE DE ESTÍMULO:	Fallos de conexión de red o energía accidentales y minúsculos
	MEDIO AMBIENTE:	Posible desconexión temporal debido a un cambio o actualización en la red actual
	ARTEFACTOS:	Sistema de comunicación entre sistema central y sistema de clientes, controlador del servidor
	RESPUESTA:	El sistema guarda las consultas pendientes para ser reinventadas automáticamente cuando la conexión se restablezca.
	MEDIDA DE LA RESPUESTA:	1 minuto de espera para posible reconexión
PREGUNTAS/ Aspectos relacionados:		

ESCENARIO QAW:		El servidor, debido a una falla inesperada, deja de estar disponible por un tiempo.
META DE NEGOCIO:		Garantizar la continuidad mínima de operaciones y reducir el impacto en los usuarios mientras el servidor se restablece.
ELEM	ATRIBUTO DE CALIDAD:	Disponibilidad, Recuperación ante fallos, Escalabilidad.
	ESTÍMULO:	Falla del servidor por problemas internos, o saturación de tráfico.

E N T O S	FUENTE DE ESTÍMULO:	Controlador del servidor
	MEDIO AMBIENTE:	El servidor se ve sobrecargado o presenta un fallo inusual que lo lleva a dejar de funcionar
	ARTEFACTOS:	Sistemas de comunicaciones, controlador del servidor, base de datos
	RESPUESTA:	El sistema detecta la indisponibilidad del servidor central y redirige las consultas a un servidor de respaldo (si existe).
	MEDIDA DE LA RESPUESTA:	Redirigir las consultas al otro servidor en un lapso menor a 10 segundos
PREGUNTAS/ Aspectos relacionados:		

ESCENARIO QAW:		Durante la ejecución de las consultas, se debe actualizar el sistema sin interrumpir las consultas en curso.
META DE NEGOCIO:		Evitar la pérdida de consultas o minimizar el impacto de las actualizaciones en los usuarios.
E L E M E N T O S	ATRIBUTO DE CALIDAD:	Disponibilidad, Modificabilidad.
	ESTÍMULO:	Aplicación de una actualización crítica durante el uso del sistema.
	FUENTE DE ESTÍMULO:	Tiempo y personal de mantenimiento
	MEDIO AMBIENTE:	Necesidad de un cambio crucial en el sistema para evitar otros problemas.
	ARTEFACTOS:	Base de datos, Sistema central, Comunicador entre sistema central y clientes
	RESPUESTA:	El sistema permanece funcional durante la actualización sin detener su ejecución
	MEDIDA DE LA RESPUESTA:	Afectar la latencia respecto a las respuestas a coste de poder implementar los cambios en el sistema sin detenerlo
PREGUNTAS/ Aspectos relacionados:		

para realizar una consulta a detalle de los casos de QAW por favor dirigirse al anexo “qaw-proyectosw4”

Comparación y análisis de patrones de diseño arquitectónico

A continuación se comparan únicamente los patrones adicionales y el patrón de distribución pues los otros patrones fueron directamente solicitados por el stakeholder y no existe la posibilidad de cambiarlos.

Comparación Patrón adicional (patrón de direccionamiento)

Criterio	Load Balancer	Gateway	Service Locator	Proxy	Broker
Descripción	Redistribuye solicitudes entre servidores según una estrategia predefinida.	Punto único de entrada que enruta solicitudes según rutas o lógica.	Encuentra y conecta servicios en un sistema distribuido.	Intermediario que reenvía solicitudes entre cliente y servidor.	Intermediario que distribuye solicitudes dinámicamente a múltiples servidores.
Direcciona miento	Básico: Redirige según estrategias como round robin o menor carga.	Centralizado: Basado en rutas o lógica definida para APIs o servicios.	Ofrece localización de servicios, pero no distribuye carga.	Básico: Redirige solicitudes con reglas simples.	Avanzado: Gestiona la distribución dinámica según criterios específicos (carga, disponibilidad, tipo de tarea).
Tolerancia a fallos	Alta: Redirige automáticamente si un servidor falla.	Alta: Puede redirigir solicitudes si un backend no responde.	Moderada: Encuentra servicios disponibles, pero no gestiona fallos en tiempo real.	Moderada: Depende de configuraciones adicionales para manejar fallos.	Alta: Si un servidor falla, redirige las solicitudes a otros disponibles automáticamente.
Balanceo de carga	Excelente: Optimizado para distribuir solicitudes equitativamente.	Moderado: Enfocado en enrutar solicitudes, no en balancear carga.	No aplica: No gestiona distribución activa de solicitudes.	Básico: Puede reenviar solicitudes pero no optimiza la distribución.	Excelente: Integra balanceo dinámico según criterios avanzados.

Escalabilidad	Alta: Fácil de añadir servidores adicionales.	Moderada: Escalar implica ajustar configuraciones de rutas.	Alta: Permite localizar servicios en sistemas muy grandes.	Moderada: Escalar depende de la lógica de reenvío.	Alta: Escalar servidores es fácil y el Broker se adapta dinámicamente.
Eficiencia de recursos	Alta: Usa todos los servidores activos para distribuir carga.	Moderada: Requiere lógica extra para optimizar recursos.	Alta: Optimiza la localización, pero no gestiona solicitudes.	Baja: Puede desperdiciar recursos al no optimizar carga.	Alta: Los recursos se distribuyen dinámicamente según disponibilidad.
Flexibilidad	Baja: Limitado a estrategias predefinidas de redirección.	Moderada: Flexibilidad en rutas, pero complejidad en gestión.	Alta: Facilita la integración de servicios, pero no gestiona carga.	Baja: Reglas simples y menos adaptable para sistemas grandes.	Alta: Flexible para múltiples criterios (disponibilidad, carga, prioridad).
Idoneidad para consultas concurrentes	Alta: Maneja múltiples solicitudes simultáneamente.	Moderada: Maneja concurrencia, pero se centra en enrutar.	Baja: No maneja consultas concurrentes activamente.	Baja: Limitado a reglas básicas de reenvío.	Excelente: Diseñado para manejar concurrencia y distribuir tareas dinámicamente.
Costo de implementación	Moderado: Uso de herramientas estándar (NGINX, HAProxy).	Moderado: Requiere lógica específica para rutas y configuraciones.	Bajo: Fácil de implementar, pero requiere registro inicial de servicios.	Moderado: Configuraciones simples de reglas y reenvío.	Moderado: Requiere lógica para monitorear, distribuir y manejar fallos.

Criterio	Load Balancer	Gateway	Service Locator	Proxy	Broker

Direccionamiento avanzado	**3**	**4**	**3**	**2**	**4**
Tolerancia a fallos	**4**	**4**	**2**	**3**	**2**
Balanceo de carga	**5**	**3**	**1**	**2**	**5**
Escalabilidad	**4**	**3**	**4**	**3**	**5**
Eficiencia de recursos	**4**	**3**	**3**	**2**	**4**
Flexibilidad	**3**	**4**	**5**	**2**	**5**
Idoneidad para concurrencia	**5**	**4**	**2**	**2**	**4**
Costo de implementación	**4**	**4**	**5**	**4**	**3**

Comparación de Patrones de distribución

Criterio	**Sender Released**	**Producer-Consumer**	**ThreadPool**	**Separable Dependency**	**Master-Worker**	**Thread Pool**
Descripción	El sistema libera las tareas a medida que las solicita el cliente, sin necesidad de coordinación central.	Un productor genera tareas, que luego son consumidas por trabajadores.	Un pool de hilos es utilizado para ejecutar múltiples tareas en paralelo.	Las dependencias son separables, lo que permite dividir una tarea en partes manejables.	Un maestro divide una tarea en subtareas y las distribuye entre varios trabajadores.	Un conjunto de hilos ejecuta tareas en paralelo, gestionadas por un pool.

Tolerancia a fallos	Moderada. Si un trabajador falla, las tareas pueden reanudarse, pero no hay un sistema centralizado de gestión de fallos.	Moderada. Si un consumidor falla, las tareas pueden volver a la cola para ser procesadas por otro consumidor.	Baja. Si un hilo falla, el trabajo podría perderse si no se gestiona adecuadamente.	Alta. Las tareas son independientes y pueden reprogramarse si hay fallos.	Alta. Si un trabajador falla, el maestro puede reasignar las subtareas a otros trabajadores disponibles.	Baja. La falla de un hilo podría afectar el procesamiento de tareas si no se gestiona adecuadamente.
Escalabilidad	Moderada. Se pueden añadir más productores, pero el sistema no siempre se adapta automáticamente a cargas crecientes.	Alta. Se pueden añadir más consumidores para manejar más tareas simultáneamente.	Alta. Se puede ajustar el tamaño del pool para manejar más tareas concurrentes.	Alta. Las dependencias se pueden dividir fácilmente en tareas más pequeñas.	Excelente. Los trabajadores pueden ser añadidos o eliminados dinámicamente para manejar más carga.	Alta. El tamaño del pool de hilos puede ajustarse según la demanda.
Balanceo de carga	Baja. No hay un mecanismo explícito para balancear las tareas entre los productores y los consumidores.	Moderado. El balanceo de carga depende de cómo se gestionan las tareas en la cola de consumidores.	Moderado. El balanceo de carga depende del número de hilos y cómo se asignan las tareas.	Moderado. Se distribuyen tareas independientes, pero el balanceo no es automático.	Excelente. El maestro distribuye las subtareas entre los trabajadores de manera equilibrada.	Moderado. El trabajo se distribuye entre los hilos, pero no siempre se hace de forma equilibrada.
Eficiencia de recursos	Baja. No se gestionan los recursos de manera centralizada, lo que puede generar ineficiencias en el procesamiento.	Alta. Los consumidores trabajan independientemente, maximizando el uso de los recursos.	Alta. Los hilos están diseñados para ejecutarse concurrentemente, maximizando el uso de CPU.	Alta. Las tareas son independientes y se pueden ejecutar en paralelo.	Excelente. Los trabajadores pueden ser asignados según la carga de trabajo, optimizando el uso de recursos.	Alta. El pool de hilos maximiza el uso de los recursos de CPU y memoria.

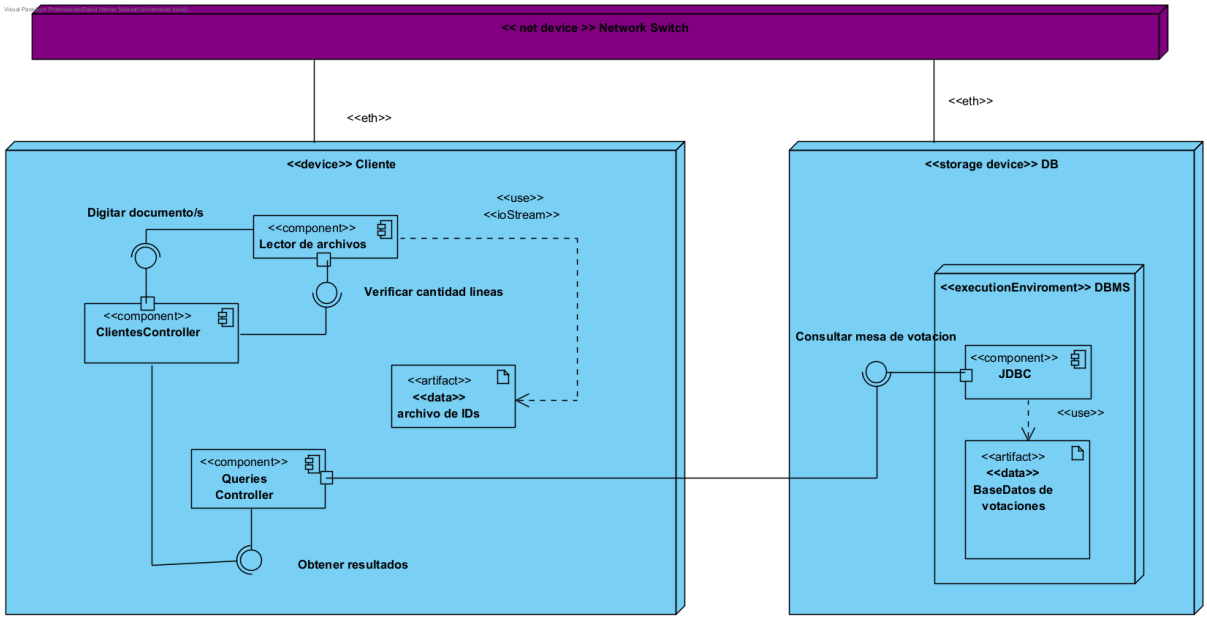
Flexibilidad	Baja. El sistema está diseñado para una liberación de tareas sin mucha flexibilidad.	Moderada. Los consumidores pueden ser cambiados, pero no hay mucha flexibilidad en el flujo de trabajo.	Moderada. El número de hilos puede ajustarse, pero no hay mucha flexibilidad en la asignación de tareas.	Alta. Las dependencias pueden separarse de manera flexible según los requisitos del sistema.	Alta. El maestro puede reorganizar o redistribuir las subtareas según sea necesario.	Moderada. Los hilos son fijos, pero el tamaño del pool puede ajustarse según la carga.
Idoneidad para consultas concurrentes	Baja. No hay una gestión eficiente para manejar múltiples consultas simultáneamente.	Moderada. Aunque los consumidores pueden manejar varias consultas, el balanceo y coordinación no siempre son eficientes.	Alta. Los hilos pueden procesar múltiples consultas concurrentemente, pero no siempre se gestionan de forma eficiente.	Alta. Las tareas son fácilmente separables y se pueden ejecutar en paralelo.	Excelente. El maestro puede gestionar grandes volúmenes de consultas distribuidas entre varios trabajadores concurrentes.	Alta. El pool de hilos permite manejar múltiples consultas, pero la asignación de trabajo puede no ser óptima.
Complejidad de implementación	Baja. Implementar el patrón es simple, pero la gestión de tareas puede ser limitada.	Moderada. Se necesita gestionar correctamente la cola de tareas y los consumidores.	Baja. Implementar un pool de hilos es sencillo, pero la gestión de tareas puede ser complicada.	Moderada. Se requiere lógica para dividir las tareas y gestionar las dependencias.	Alta. La lógica para dividir tareas, gestionar los trabajadores y combinar resultados requiere más complejidad.	Baja. La implementación de un pool de hilos es relativamente fácil, pero la coordinación puede ser más difícil.

Criterio	**Sender Released**	**Producer-Consumer**	**ThreadPool**	**Separable Dependency**	**Master-Worker**	**Thread Pool**
Tolerancia a fallos	2	3	3	4	4	3

Escalabilidad	2	3	4	4	5	4
Balanceo de carga	1	3	3	3	5	3
Eficiencia de recursos	2	4	4	4	3	4
Flexibilidad	2	3	3	4	5	4
Idoneidad para consultas concurrentes	2	3	4	4	5	4
Complejidad de implementación	4	3	3	4	2	3

Plan de implementación diagrama de deployment.

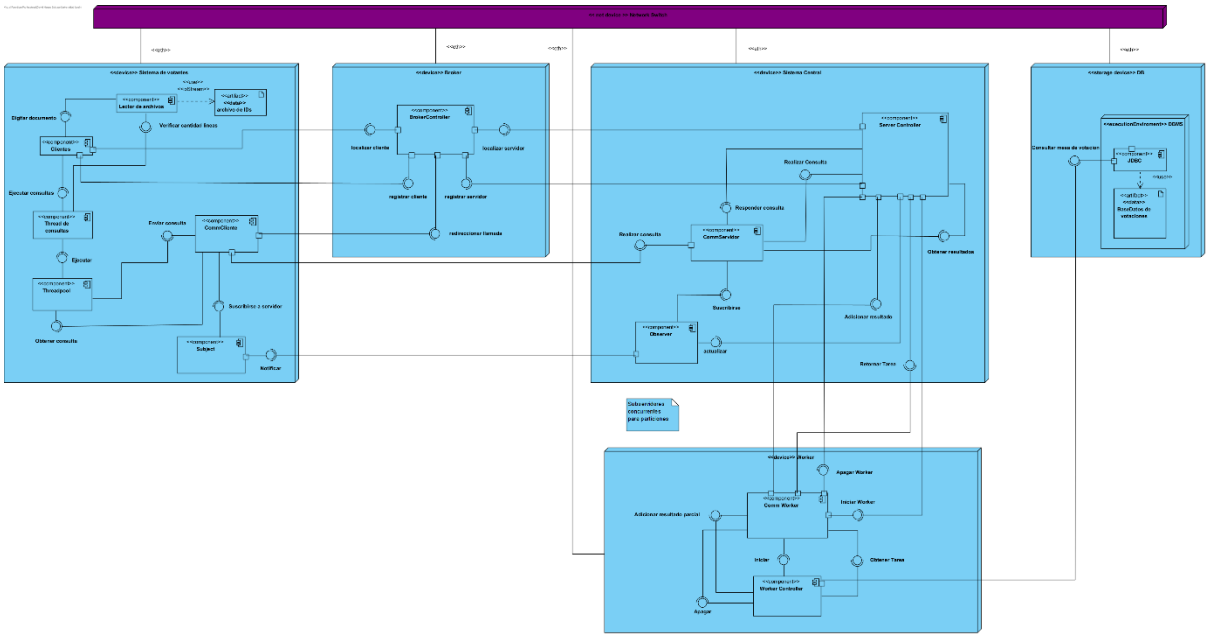
Diagrama Deployment Monolitico



Para la realización de la primera versión del diagrama de despliegue del sistema tuvimos en cuenta la manera básica del funcionamiento, manteniendo todos los componentes dentro de

un solo dispositivo, teniendo un único sistema de conexión dirigido a la base de datos para poder realizar las consultas.

Diagrama Deployment Distribuido



Para la construcción del diagrama de despliegue distribuido, se seleccionaron los patrones arquitectónicos más adecuados para satisfacer los requisitos y atributos de calidad del proyecto, considerando su impacto en los diferentes componentes del sistema. Los patrones escogidos fueron Broker, Observer, Master-Worker y ThreadPool, con un orden de implementación basado en el impacto que cada patrón tendría en los componentes del sistema y en el diseño general.

El orden de implementación fue el siguiente:

Broker: Este patrón se implementó primero debido a su función clave como intermediario en la comunicación entre clientes y servidores. Al establecer el Broker desde el inicio, se garantizó una distribución eficiente de las solicitudes y una comunicación flexible entre los diferentes componentes distribuidos del sistema.

Observer: Una vez configurado el Broker, se introdujo el patrón Observer para gestionar la notificación de eventos, especialmente la distribución de cargas y la respuesta a cambios en el estado del servidor. Este patrón fue implementado en segundo lugar, ya que permite una interacción dinámica entre el servidor y los clientes sin comprometer el flujo de trabajo principal.

Master-Worker: Con el Broker y el Observer en su lugar, se implementó el patrón Master-Worker para mejorar la escalabilidad y eficiencia del procesamiento de tareas. Este patrón permite dividir las consultas en subtareas, distribuyendo las entre múltiples trabajadores, lo que optimiza el rendimiento en sistemas con alta concurrencia. Su implementación se posicionó en tercer lugar, ya que depende de una gestión adecuada de las tareas y la capacidad de los servidores distribuidos.

ThreadPool: Finalmente, se implementó el patrón ThreadPool para optimizar la ejecución concurrente de tareas dentro de cada cliente, maximizando el uso de los recursos disponibles. Este patrón fue implementado en último lugar, ya que su propósito es mejorar la eficiencia interna del sistema, facilitando la ejecución paralela de tareas sin interferir con los patrones previos.

Este orden de implementación asegura que cada patrón contribuya de manera incremental al diseño del sistema, considerando su impacto tanto en los componentes individuales como en la interacción entre ellos, y optimizando el rendimiento global.

Diseño de experimentos y pruebas.

Respecto al diseño experimentos se medirán los tiempos y la tolerancia a fallos cuando se tienen altos volúmenes de datos para esto se realizará la experimentación de la siguiente manera:

- se codificara una versión del código básica monolítica ya que usaremos la misma como medida de comparación para cuando se implementen los patrones y se realice el código de manera distribuida
- Se codifica una segunda versión del código distribuida y que aplicará los patrones anteriormente mencionados en el documento se medirá su punto de tolerancia y sus tiempos.

Los datos esperados es que en la versión distribuida se tenga un mayor margen de tolerancia con unos tiempos similares a la versión monolítica. Adicionalmente se entregará como evidencia de la experimentación una serie de archivos tipo tabla de cálculo donde se detallan los diferentes experimentos realizados y sus respectivos resultados.

Respecto al diseño de pruebas nos centraremos en las pruebas de rendimiento pues según el stakeholder es un factor crítico de su negocio por ende las medidas utilizadas serán de tiempo, adicionalmente con estas pruebas se usará los logs como modo de verificación.

ID	**Caso de prueba**	**Descripción**	**Carga simulada**	**Métrica esperada**
PR1	Latencia con consultas simultáneas	Ejecutar consultas desde 3 clientes concurrentes.	3 consultas	Tiempo de respuesta por consulta ≤ 1 segundo.
PR2	Escalabilidad con clientes adicionales	Agregar clientes hasta alcanzar 6 conexiones.	6 conexiones	El sistema responde sin fallos manteniendo la conectividad a los 6 clientes

PR3	Procesamiento de tareas grandes	Enviar archivo con 100,000 cédulas para consulta.	Archivo grande	Consultas procesadas y distribuidas correctamente entre los workers en menos de 10 minutos.

Foto Prueba 1

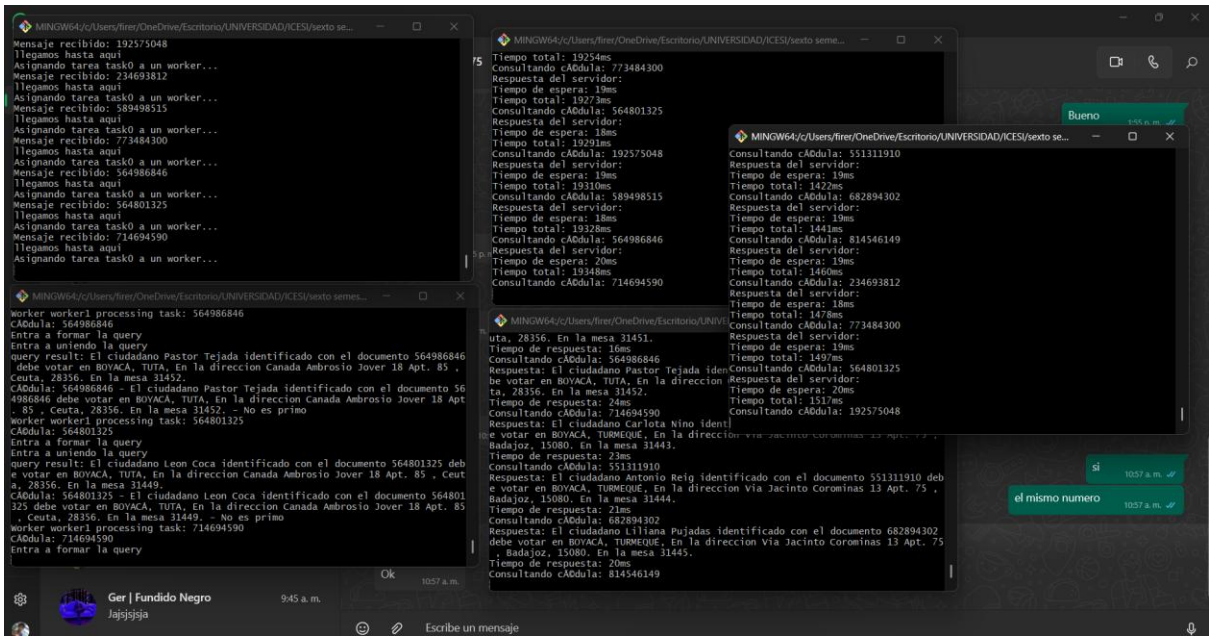


Foto Prueba 2

No cupieron en la captura

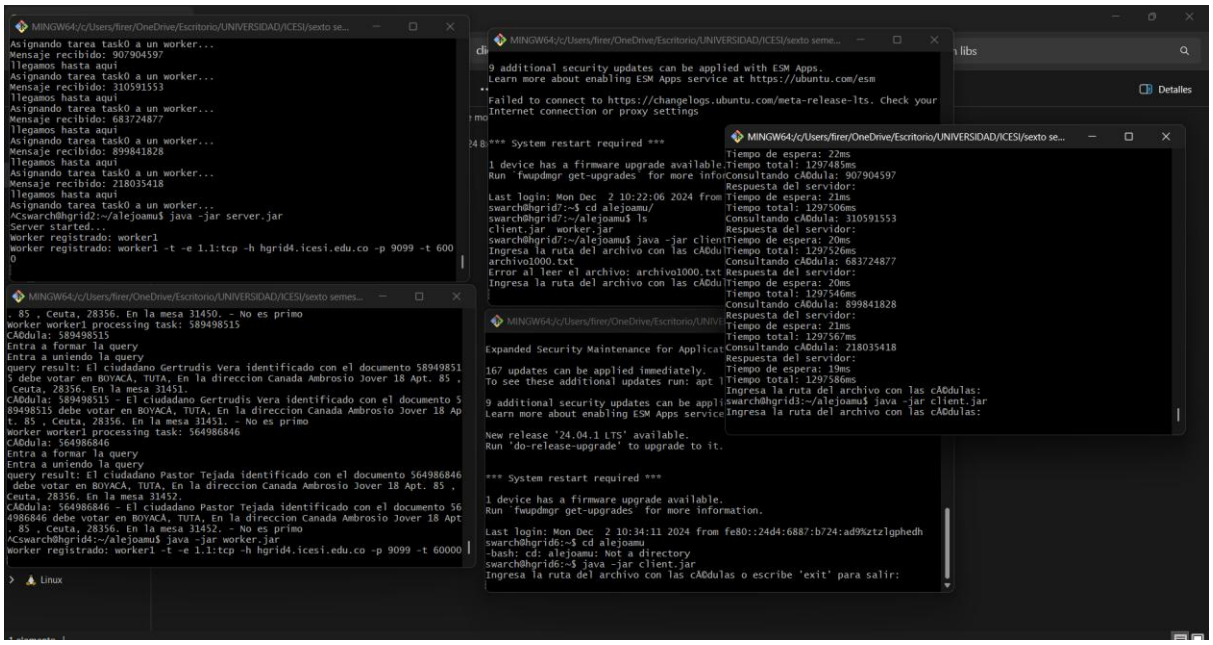
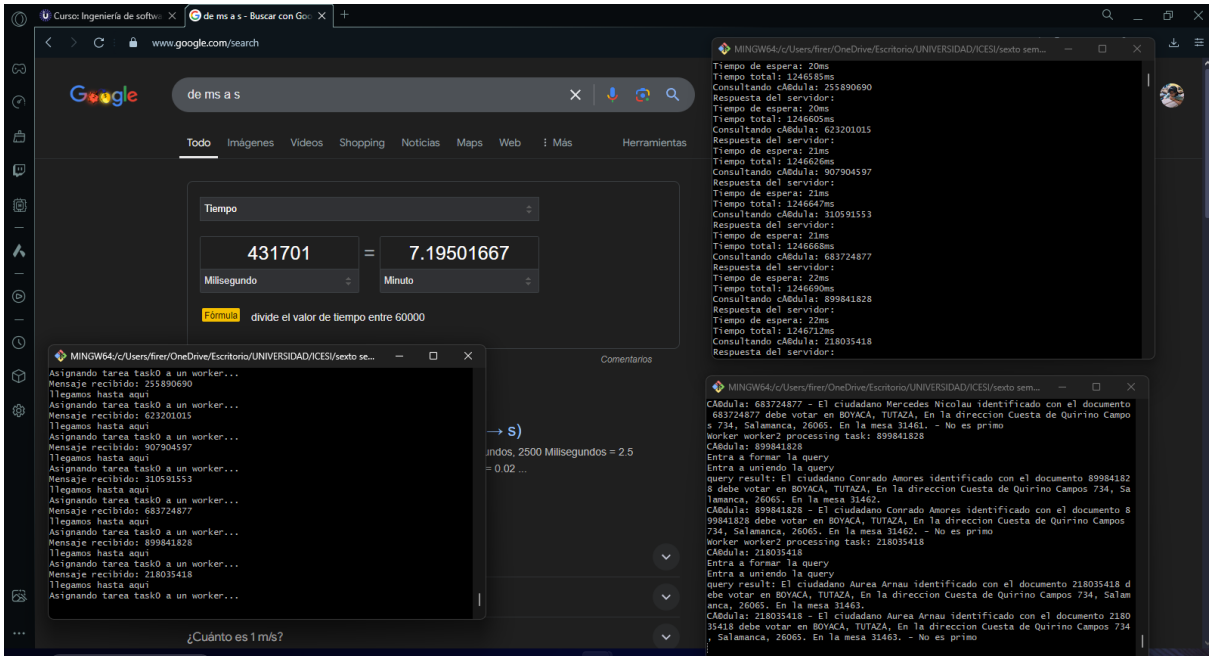


Foto Prueba 3


Curso: Ingeniería de softwa
de ms a s - Buscar con Goo
www.google.com/search
Google
de ms a s
Todo
Imágenes
Videos
Shopping
Noticias
Maps
Web
Más
Herramientas
Tiempo
431701
=
7.19501667
Milisegundo
Minuto
Fórmula
divide el valor de tiempo entre 60000
Comentarios
→ s)
ndos, 2500 Milisegundos = 2.5
= 0.02 ...
¿Cuánto es 1 m/s?
MINGW64:/c/Users/firer/OneDrive/Escritorio/UNIVERSIDAD/ICESI/sexto se...
Asignando tarea task0 a un worker...
Mensaje recibido: 255890690
Llegamos hasta aqui
Asignando tarea task0 a un worker...
Mensaje recibido: 623201015
Llegamos hasta aqui
Asignando tarea task0 a un worker...
Mensaje recibido: 907904597
Llegamos hasta aqui
Asignando tarea task0 a un worker...
Mensaje recibido: 310591553
Llegamos hasta aqui
Asignando tarea task0 a un worker...
Mensaje recibido: 683724877
Llegamos hasta aqui
Asignando tarea task0 a un worker...
Mensaje recibido: 899841828
Llegamos hasta aqui
Asignando tarea task0 a un worker...
Mensaje recibido: 218035418
Llegamos hasta aqui
Asignando tarea task0 a un worker...
MINGW64:/c/Users/firer/OneDrive/Escritorio/UNIVERSIDAD/ICESI/sexto sem...
Tiempo de espera: 20ms
Tiempo total: 1246585ms
Consultando cÃ©dula: 255890690
Respuesta del servidor:
Tiempo de espera: 20ms
Tiempo total: 1246605ms
Consultando cÃ©dula: 623201015
Respuesta del servidor:
Tiempo de espera: 21ms
Tiempo total: 1246626ms
Consultando cÃ©dula: 907904597
Respuesta del servidor:
Tiempo de espera: 21ms
Tiempo total: 1246647ms
Consultando cÃ©dula: 310591553
Respuesta del servidor:
Tiempo de espera: 21ms
Tiempo total: 1246668ms
Consultando cÃ©dula: 683724877
Respuesta del servidor:
Tiempo de espera: 22ms
Tiempo total: 1246690ms
Consultando cÃ©dula: 899841828
Respuesta del servidor:
Tiempo de espera: 22ms
Tiempo total: 1246712ms
Consultando cÃ©dula: 218035418
Respuesta del servidor:
MINGW64:/c/Users/firer/OneDrive/Escritorio/UNIVERSIDAD/ICESI/sexto sem...
CÃ©dula: 683724877 - El ciudadano Mercedes Nicolau identificado con el documento 683724877 debe votar en BOYACA, TUTAZA, En la direccion Cuesta de Quirino Campos 734, Salamanca, 26065. En la mesa 31461. - No es primo
Worker worker2 processing task: 899841828
CÃ©dula: 899841828
Entra a formar la query
Entra a uniendo la query
query result: El ciudadano Conrado Amores identificado con el documento 899841828 debe votar en BOYACA, TUTAZA, En la direccion Cuesta de Quirino Campos 734, Salamanca, 26065. En la mesa 31462.
CÃ©dula: 899841828 - El ciudadano Conrado Amores identificado con el documento 899841828 debe votar en BOYACA, TUTAZA, En la direccion Cuesta de Quirino Campos 734, Salamanca, 26065. En la mesa 31462. - No es primo
Worker worker2 processing task: 218035418
CÃ©dula: 218035418
Entra a formar la query
Entra a uniendo la query
query result: El ciudadano Aurea Arnau identificado con el documento 218035418 debe votar en BOYACA, TUTAZA, En la direccion Cuesta de Quirino Campos 734, Salamanca, 26065. En la mesa 31463.
CÃ©dula: 218035418 - El ciudadano Aurea Arnau identificado con el documento 218035418 debe votar en BOYACA, TUTAZA, En la direccion Cuesta de Quirino Campos 734, Salamanca, 26065. En la mesa 31463. - No es primo