Diseño de experimento

1. Alcance

A continuación, se describe de manera detallada el Diseño del Experimento para validar que la arquitectura para realizar el voto:

Nodo de votación → Lugar de votación → (por reliable message)→ Servicio de votación (y servicio de autenticación)→ Base de datos

se cumplan estrictamente los requisitos de confiabilidad (100% de votos registrados) y unicidad (ningún voto contado más de una vez). Adicionalmente que para la arquitectura:

Nodo de consulta → servicio de consulta → base de datos

Se cumpla con tener una alta disponibilidad y escalabilidad. Cabe la pena aclarar que no se justifica realizar un proxy cache para el último caso de uso, esto porque muy difícilmente las consultas sobre un mismo lugar de votación, se realicen secuencialmente, haciendo que se sobreescriba mucho el contenido del proxy caché. Este diseño se basa en la aclaración de que la comunicación interna dentro del Lugar de votación y los nodos individuales es confiable, pero la comunicación externa (Lugar de votación al Servicio de votación) puede no serlo.

2. Objetivos del experimento

Validar Confiabilidad (Tolerancia a Fallos en la Conexión Externa)

- Garantizar que, incluso si la conexión entre el “Lugar de votación” y el “Servicio de votación” falla intermitentemente, todos los votos generados en los Nodos de votación terminen persistidos en la Base de datos del Servicio.

Verificar Unicidad (Idempotencia y Detección de Duplicados)

- Asegurar que, sin importar la cantidad de reenvíos (por reintentos o duplicaciones accidentales), el “Servicio de votación” persista cada voto exactamente una sola vez en la tabla de votos.

Confirmar Atributos de Calidad Específicos

- Consistencia: los resultados en la Base de datos (conteos) reflejen fielmente los votos únicos sin pérdidas ni duplicaciones.
- Escalabilidad: El proceso de realizar votación y consulta de lugar de votación debe ser escalable para que no hayan problemas por la alta cantidad de usuarios concurrentes.
- Disponibilidad: El proceso de realizar votación y el de consultar lugar de votación debe ser altamente disponible, para que los ciudadanos puedan votar.

3. Explicación de la arquitectura

3.1 Nodo de votación (CLI local)

El Nodo de Votación es una pequeña aplicación de consola instalada en cada terminal físico de mesa.

- Validación básica de formato y rango de cédula.
- Ubicado en cada máquina de votación (por ejemplo, un terminal de la sala de cómputo).
- Provee una interfaz de línea de comandos para que el operador o ciudadano ingrese su número de documento y seleccione el candidato.
- Construye internamente un objeto de voto que incluye:
    - voterId (String)
    - candidateId (String)

Se comunica de forma directa con el “Lugar de votación” para enviar el voto.

La comunicación entre el Nodo y el Lugar se considera 100 % confiable.

3.2 Lugar de votación (Servidor ICE local)

Cada estación (mesa) ejecuta un proceso que expone la interfaz ICE VotingSite con el método sendVote(Vote vote). Se considera confiable la comunicación pues está en la misma red LAN que el nodo de votación.

Funciones principales:

- Lógica de envío confiable (Reliable Messaging) hacia el Servicio de votación:
    - Cada vez que llega sendVote(vote), el VotingSiteController encola ese Vote en un BatchingController.
- El BatchingController agrupa votos en un List<Vote> cuando se cumple cualquiera de estos dos límites:
    - Han transcurrido 500 ms desde que llegó el primer voto del lote.
    - Se acumulan 1000 votos en la cola.
- Se envía el lote por reliable message al servicio de votación, se reintentará el envío cada 10 segs.
- Se suscribe al servicio de control para iniciar y terminar la jornada de votación
- Se suscribe, antes de la jornada, al servicio de candidatos para obtener los datos de los candidatos..

3.3 Servicio de autenticación

Servicio ligero orientado a reglas de voto por mesa. Todas sus verificaciones las realiza contra una base de datos. Solo verifica si el ciudadano ya ha votado o no, y si está votando en la mesa indicada. Los datos de que ciudadanos pueden votar en qué mesa se cargan antes de la jornada de votación.

3.4 Servicio de candidatos

Nodo central que es publisher de los datos de los candidatos. Los suscribers son los lugares de votación.

3.5 Servicio de votación

Expone la interfaz ICE RMDestination.receiveMessage()

Recepción de cada lote para recibir los votos:

- Envía el ACK con el uuid
- Luego verifica los duplicados y la validez mediante el nodo de servicio de autenticación.

3.6 Persistencia en Base de datos

Usa un esquema relacional (H2) con una tabla Votos que incluye columnas obligatorias:

- voter_id VARCHAR(255)
- candidate_id VARCHAR(255)
- Clave primaria compuesta (voter_id, candidate_id).

Al término de la jornada, esta tabla es la única fuente de la verdad para:

- Validación de unicidad: no puede haber dos filas con el mismo (voter_id, candidate_id).
- Generación de conteos para el CSV final.

3.7 Nodo de consulta (CLI local)

Simple interfaz gráfica por consola, que permite consultar los resultados de la votación y retorna un .csv general del resultado y un .csv por mesa de votación.

3.8 Servicio de Consulta Resultados

Mediante un proxy caché, este servicio lee la base de datos para la generación del .csv general y por mesa al final de la votación.

3.9 Servicio de Consulta Lugar de votación

Se conecta a la base de datos, recibe la cédula del ciudadano y retorna el lugar de votación.

3.10 Flujo del proceso del voto

Para resumir, el flujo del voto es el siguiente:

Paso	Actor / Componente	Acción	Detalle técnico
0.1	VotingCandidatesService → Sitios/Mesas	Publica: candidates.update() Con antelación	Pub/Sub Cada Sitio persiste los candidatos
0.2	Control service → Sitios	Envía startElection.	Canal de control (pub/sub). Activa la recepción de votos.
1	Ciudadano → Nodo de Votación	Ingresa voterId y elige candidateId.	Validación de formato; lista de candidatos tomada del caché

			local.
2	Nodo → Sitio de Votación	VotingSite.sendVote() Junta varios para hacer el batch.	Red LAN (100 % confiable).
3	Sitio	Encola voto en BatchingController.	Cola en memoria.
4	Sitio → Broker	Al llegar 1000 votos *o* 500 ms, Se conecta al broker para envíar la lista de los votos	RMSource.sendMessage(batchUuid, payload).
5	Broker RM	re-intenta cada 10 s hasta ACK del Servicio de Votación. Envía el batch al servicio	El sitio espera el ACK
6	Servicio de Votación	RMDestination.receiveMessage(batch) → responde ACK al Broker.	ACK libera al Sitio; el Broker marca lote como entregado.
7	Servicio de Votación → Servicio de Autenticación	Envía (voterId,lugarId,mesaId) por voto.	
8	Autenticación → Redis	Consulta la validez del voto.	
9	Autenticación → Servicio de Votación	Devuelve el resultado	
10	Servicio de Votación	Filtra no-autorizados; inserta válidos con INSERT … ON CONFLICT DO NOTHING en Votes.	PK (voter_id,candidate_id) impide duplicados reales.
11	Servicio de Votación → Redis	Actualiza los datos para marcar el ciudadano como que ya votó	Evita doble voto en futuras validaciones.
…	…	…	…
12	Admin → Sitios	Envía endElection.	Sitios detienen sendVote.
13	Servicio de Votación	Genera resume.csv global + mesa_<id>.csv y los expone al Servicio de Consulta de resultados.	Lectura directa de Votes.

3.11 Flujo de la consulta del lugar de votación

El flujo de la consulta del lugar de votación es el siguiente:

Paso	Actor / Componente	Acción
1	Ciudadano → Nodo de Consulta	Introduce voterId y pulsa “Consultar”.
2	Nodo → Servicio de Consulta	Recibe la cédula
3	Servicio de Consulta → BD	Consulta el resultado en la base de datos y la retorna

4. Metodología de prueba

4.1 Entorno para pruebas de votación:

Rol/función	Nº de PC	Detalle de despliegue
Nodos de votación (CLI)	6	Se usarán o bien otros computadores u otros hilos para aumentar la cantidad de nodos.
Sitios de votación / Mesas	2	Cada Sitio atiende una mesa física (mesa 1, mesa 2) y actúa como cliente de Reliable Messaging.
Broker Reliable Messaging	1	ICEgrid
Servicios de votación	2	Cada instancia conecta a la misma BD.
Servicios de autenticación	2	Réplicas idénticas
bd	1	Claves
PostgreSQL (BD de votos)	1	Tabla de votos

Se usarán 15 pc’s

4.1.1 Variables experimentales

- C (clientes de votación) = 6, 12, 18 hilos totales.
- V (votos emitidos) = 6000; 12000; 18000.
- T (duración de prueba) = 10 segundos y 20 segundos.

Esto genera 3 × 3 × 2 = 18 ejecuciones; el orden se aleatoriza.

4.1.2 Pasos

- Precargar la bd
- Iniciar Broker, servicios de votación, los servicios de autenticación, la base de datos y sitios de votación.
- Cargar con los candidatos a los sitios de votación.
- Se realizan los votos: se reparten equitativamente los votos emitidos totales (V)(6000, 12000, 18000) entre los hilos que se van a ocupar en votar (C)(6, 12, 18 hilos), estos hilos mandan los votos uniformemente por el tiempo de ejecución (T)(10 segundos, 20 segundos).
- Se cierra la elección
- Se sacan las siguientes métricas:

Métrica	**Cómo se calcula / dónde se lee**
T_vote_total	Marca de tiempo primero ↔ último ACK.
Throughput_vote (vps)	V / T_vote_total. (Idealmente que cumpla los votos por segundo mencionados: 1777/s)
Distribución de carga	% lotes atendidos por Servicio 1 vs Servicio 2 (Una diferencia menor al 10%).
Duplicados	SELECT COUNT(*)-COUNT(DISTINCT voter_id,candidate_id) FROM Votes; (esperado 0).

Para un T = 10 segundos y un V = 18000 se obtiene un throughput de 1800, superior a los 1777 que se piden.

4.2 Entorno para pruebas de consulta de lugar de votación:

Rol / función	Nº de PC	Detalle de despliegue
Clientes de consulta (CLI/Web)	5	Se usarán o bien otros computadores u otros hilos para aumentar la cantidad de nodos.
Servicio de consulta	3	API ICE que lee directamente de Redis.
Redis	1	

Se usarán 9 pc's.

4.2.1 Variables experimentales

- C (consultores de su lugar de votación) = 5, 10, 15 hilos totales.
- Q (Consultas totales) = 3000; 15000; 30000.
- T (duración de prueba) = 10 segundos y 20 segundos.

3 × 3 × 2 = 18 ejecuciones, orden aleatorizado.

4.2.2 Pasos:

- Precargar Redis: Ingresar 30000 registros allowlist y sus correspondientes mesaInfo y eliminar cualquier clave de prueba previa.
- Arrancar servicios
- Iniciar Redis y el Servicio de consulta.
- Lanzar (5, 10, 15) procesos cliente; cada uno genera (Q / (5, 10 o 15)) consultas durante el intervalo T. Las consultas se distribuyen de forma uniforme; 5 % utilizan documentos inexistentes para forzar errores porque no se encuentre.
- Detener los procesos
- Obtener las siguientes métricas:

Métrica	Cómo se calcula / dónde se lee
T_query_total	Marca de tiempo entre la primera y la última respuesta.
Throughput_query (qps)	Q / T_query_total (objetivo ≥ 2 666 qps en nivel C = 30, Q = 30000, T = 10 s).
pico de reconexión Redis	Nº de segundos sin servicio durante la caída planeada.

Para un Q = 30000 y un T = 10 segundos se obtiene un throughput de 3000 por segundo, superior a los 2777 que se piden.

4.3 Logs:

4.3.1 Nodo de votación (CLI):

- [INFO] [VOTE SENDED]: <voterId> <candidateId> <mesaId>

4.3.2 Lugar de votación:

- [INFO] Voting Site is running
- [INFO] Processing batch of <n> votes
- [INFO] [RELIABLE MESSAGING] Starting Reliable Server
- [INFO] [RELIABLE MESSAGING] Message sent: <uuid_del_lote>
- [INFO] [RELIABLE MESSAGING] Cicle <número_de_ciclo> completed. Pending messages: <N>
- [ERROR] [RELIABLE MESSAGING] <detalle_error> - msg: <uuid_del_lote>

4.3.3 Servicio de votación:

- [INFO] Voting Service is running
- [ERROR] Error initializing communicator: <detalle_error>

- [INFO] Received batch with size: <tamaño_del_lote_de_votos>
- [INFO] Vote Batch registered successfully
- [WARNING] Duplicated vote detected for voter: <voterId> and candidate: <candidateId>
- [ERROR] [H2] Failed saving vote: <detalle_error>

4.3.4 Inicio y cierre de jornada:

- START_CLOSE_ELECTION
- END_CLOSE_ELECTION

4.3.5 Broker

- [BROK] INFO QUEUED uuid=<UUID_Lote> size=<N> ts=<ISO8601>
- [BROK] INFO DELIVERED uuid=<UUID_Lote> svc=<VoteSrv-ID>
- [BROK] WARN RETRY uuid=<UUID_Lote> svc=<VoteSrv-ID> attempt=<#>
- [BROK] INFO ACK_OK uuid=<UUID_Lote>

4.3.6 Servicio de autenticación:

- [AUTH] INFO STARTUP redis=<host:port>
- [AUTH] INFO OK voter=<voterId> mesa=<mesaId> lugar=<lugarId>
- [AUTH] WARN FORBIDDEN voter=<voterId> mesa=<mesaId> lugar=<lugarId>
- [AUTH] WARN CONFLICT voter=<voterId> (duplicate vote)
- [AUTH] ERROR REDIS_LOST reconnecting…

4.3.7 Servicio de consulta:

- [CONS] INFO STARTUP build=<ver> redis=<host:port>
- [CONS] INFO LOOKUP_OK voter=<voterId> mesa=<mesaId> lugar=<lugarId> t=<ms>
- [CONS] WARN LOOKUP_404 voter=<voterId> t=<ms>
- [CONS] ERROR REDIS_LOST reconnecting...
- [CONS] INFO REDIS_UP downtime=<s>

4.3.8 Cliente de consulta:

- [CLIS] INFO QUERY_SENT id=<seq> voter=<voterId>
- [CLIS] INFO QUERY_OK id=<seq> voter=<voterId> t=<ms>
- [CLIS] WARN QUERY_404 id=<seq> voter=<voterId> t=<ms>
- [CLIS] ERROR QUERY_FAIL id=<seq> voter=<voterId> reason=<Timeout|503>

4.4 Consultas SQL para verificar:

- Conteo total de votos persistidos:

SELECT COUNT(*) AS total_votos FROM Votes;

- Conteo de votos únicos por documento y mesa

```
SELECT COUNT(DISTINCT voter_id, candidate_id) AS votos_unicos, COUNT(*) AS total_votos
FROM Votes;
```

- Verificación de duplicados (no debe devolver filas)

```
SELECT voter_id, candidate_id, COUNT(*) AS ocurrencias
FROM Votes
GROUP BY voter_id, candidate_id
HAVING COUNT(*) > 1;
```

5. Casos de prueba extensos

Estos casos de prueba tienen el objetivo de verificar el throughput del sistema para periodos largos, en este caso, de 30 minutos.

5.1 Caso de Votación

Se usarán 6 nodos de votación, 3 sitios de votación y 2 servicios de votación. Se enviarán un total de 3200000 votos equitativamente a lo largo de 30 minutos (1777 votos por segundo). Cada nodo de votación enviará la misma cantidad de votos de manera uniforme a lo largo de los 30 minutos.

5.1.1 Pasos:

Precargar Redis: Asegurarse de que todos los registros de ciudadanos y mesas de votación están cargados en Redis.

Iniciar Servicios:

- Iniciar el broker de Reliable Messaging, los servicios de votación y los sitios de votación.
- Configurar los nodos de votación para enviar votos.

Configuración de nodos de votación:

- Cada uno de los 6 nodos se encarga de distribuir uniformemente los votos.
- Cada nodo envía 296 votos por segundo.
- Los nodos envían votos uniformemente a lo largo de 30 minutos, cada segundo un voto.

Monitoreo:

- Durante 30 minutos, verificar el throughput de votos: 1,777 votos por segundo.
- Cada sitio de votación procesará 592 votos por segundo.
- Los servicios de votación manejarán el lote de votos en batches, enviando ACK una vez procesado el lote de 1,000 votos.

- Realizar una verificación en tiempo real de los logs de los nodos de votación y sitios de votación para asegurar que el throughput se mantiene en 1,777 votos por segundo y que no haya pérdida de votos.

Finalización:

- Al término de los 30 minutos, se detienen todos los nodos y sitios de votación.
- Se realiza una consulta SQL para verificar que todos los votos fueron registrados correctamente.

5.1.2 Métricas a verificar:

- T_vote_total: Marca de tiempo entre el primer y último ACK.
- Throughput_vote (vps): 3,200,000 / 1,800 segundos = 1,777 votos por segundo.
- Latencia: Se obtiene restando el tiempo en el que se envió el último lote al tiempo en el que se proceso el último lote
- Reintentos RM: Número de intentos de reenvío durante la transmisión.
- Distribución de carga: Balance entre los 3 sitios de votación y los 2 servicios de votación.
- Duplicados: Verificación de duplicados con la consulta: SELECT COUNT(*)-COUNT(DISTINCT voter_id,candidate_id).

5.2 Caso de Consulta de Lugar de Votación

Se usarán 10 nodos de consulta y 4 servicios de consulta. Se enviarán un total de 4000000 de queries a lo largo de 30 minutos (2666 consultas por segundo). Cada nodo de consulta en total enviará 400000 peticiones por minuto.

5.2.1 Pasos:
Precargar Redis: Asegurarse de que los registros de los ciudadanos y lugares de votación estén cargados en Redis.

Iniciar Servicios:

- Iniciar el servicio de consulta y los nodos de consulta.

Configuración de nodos de consulta:

- Cada uno de los 10 nodos de consulta realizará 267 consultas por segundo a Redis.

- Las consultas deben ser distribuidas uniformemente a lo largo de los 30 minutos.
- 5% de las consultas deben ser consultas a documentos inexistentes para forzar respuestas por elemento inexistente (404).

Monitoreo:

- Durante 30 minutos, verificar el throughput de consultas: 2,666 consultas por segundo.
- Cada servicio de consulta procesará 667 consultas por segundo.
- Medir la latencia de las respuestas y asegurarse de que el tiempo de respuesta total no supere los 3 segundos.
- Realizar una verificación en tiempo real de los logs para asegurar que el throughput se mantiene en 2,666 consultas por segundo.

Finalización:

- Al término de los 30 minutos, se detienen todos los nodos de consulta.
- Se consulta el log de Redis para verificar que todas las consultas fueron procesadas correctamente.

5.2.2 Métricas a verificar:

- T_query_total: Marca de tiempo entre la primera y la última respuesta de consulta.
- Throughput_query (qps): 4,000,000 / 1,800 segundos = 2,666 consultas por segundo.
- Latencia: Se obtiene restando el tiempo en el que se envió el último lote al tiempo en el que se proceso el último lote
- Reintentos Redis: Número de reintentos debido a fallos de servicio o conexiones perdidas.
- Distribución de carga: Balance entre los 10 nodos de consulta y los 4 servicios de consulta.
- Picos de 404: Número de errores 404 generados por consultas a cédulas inexistentes.

6. Resultados

6.1 Punto de corte