U.B.A. FACULTAD DE INGENIERÍA

Departamento de Informática

Sistemas Distribuidos I 75.74

TRABAJO PRÁCTICO N° 3

Tolerancia a fallos

Curso 2020 - 2do Cuatrimestre

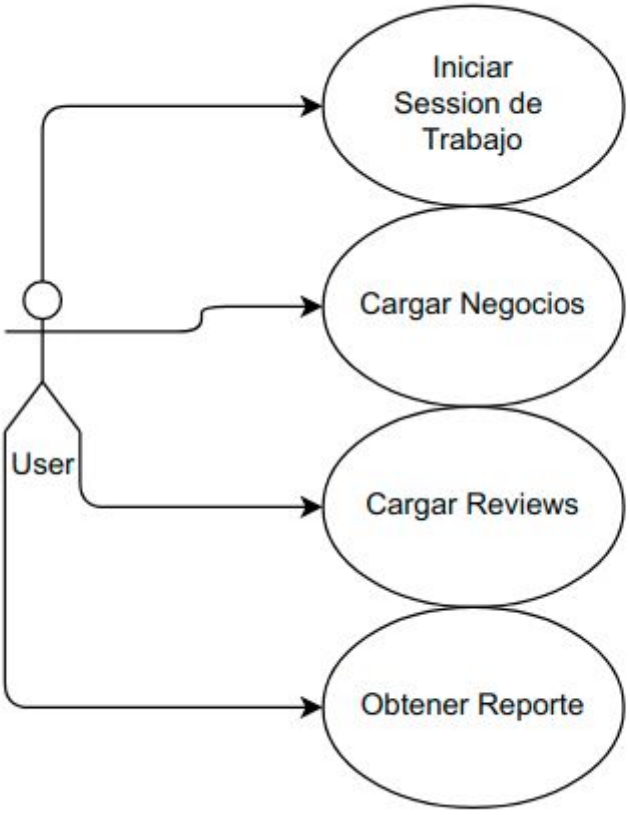
APELLIDO, Nombres	Padrón
Avigliano, Patricio Andres	98861
Cuneo, Paulo César	84840
Fecha de Aprobación :	
Calificación :	

Observaciones:

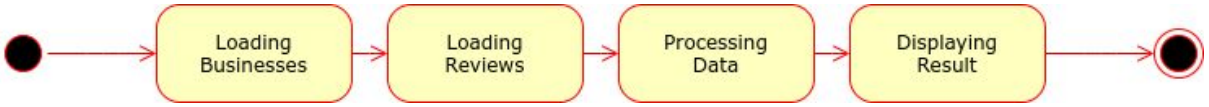
Vista Lógica:

Funcionamiento general:

El sistema permite obtener información analitica sobre las reviews de yelp, a partir de analizar un stream continue de datos. En particular existe un stream con la información de los negocios registrados en Yelp y otro stream con las revisiones de los usuario de Yelp propiamente.

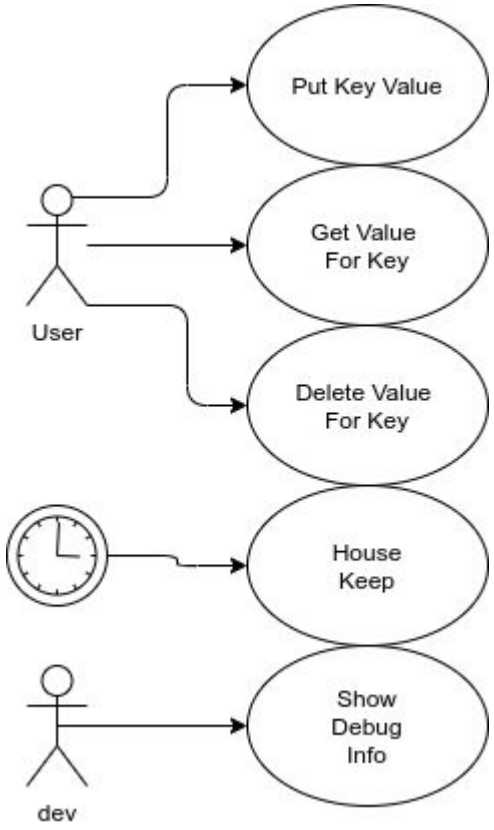


Para el modelo mental del usuario el sistema pasa por los siguiente estados.



Según la visión del usuario, el procesamiento es continuo, es decir, él puede sobre una sesión de trabajo agregar incrementalmente negocios y reviews, para luego obtener resultados parciales.

Como veremos en las siguiente vista los requerimientos técnicos nos llevarán a implementar un almacenamiento estable. El cual presenta los siguientes casos de uso.



Vista de Procesos:

Tolerancia a fallos:

Para poder asegurar que el resultado del procesamiento no va a estar afectado por la potencial caída de nodos de nuestro sistema debemos considerar los siguientes requisitos:

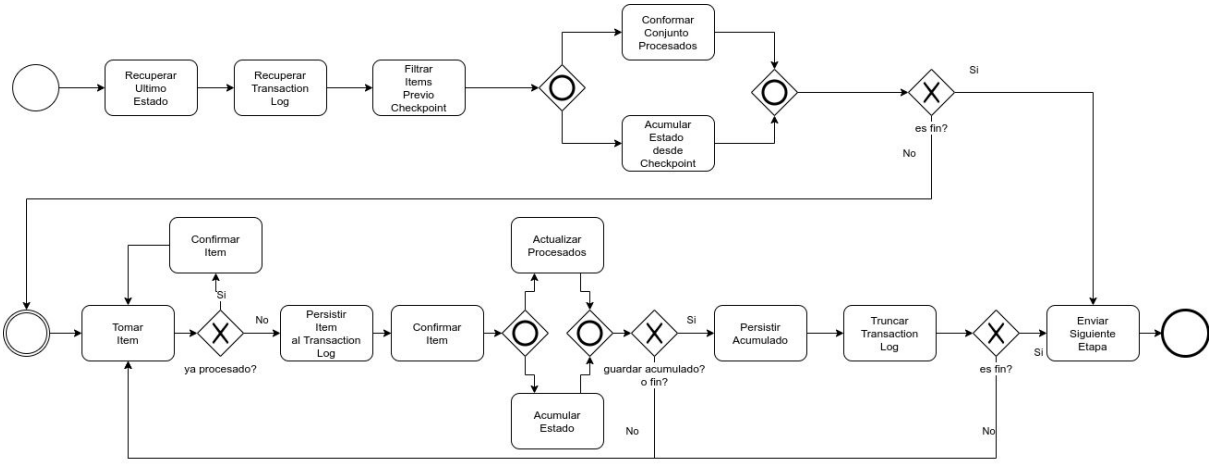
- Recuperación y persistencia
- Monitorización
- Consenso entre réplicas

A continuación explicamos las estrategias planteadas para abarcar cada uno de los requisitos mencionados

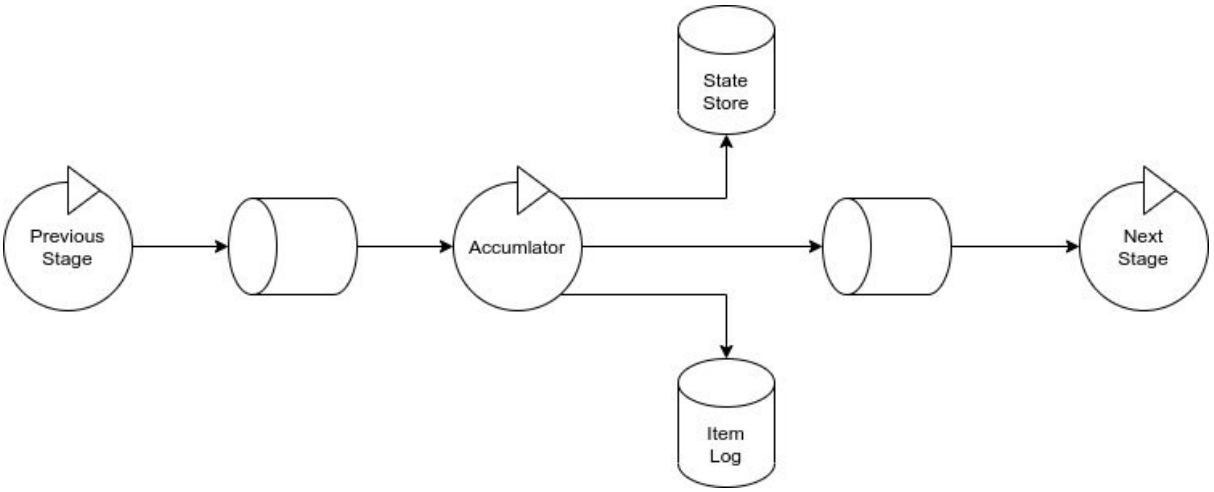
Recuperación y persistencia:

Luego de que suceda una falla el sistema de poder recuperarse y continuar procesando los datos, como si la falla no hubiera ocurrido.

Para ellos la estrategia utilizada es persistir los elementos ya procesados en un transaction log, y cada cierta cantidad de elementos tomar un snapshot del estado acumulado, y descartar los datos del transaction log previos al snapshot.
La idea es que frente a una falla tengamos todos los datos entre el checkpoint previo y la falla.
De forma tal de reconstruir el estado al momento de la falla.
El algoritmo de procesamiento sigue los pasos indicados en el diagrama de actividad que sigue:



Haciendo foco en esta parte del sistema podemos observar la interacción entre el proceso acumulador y su contexto de operación:



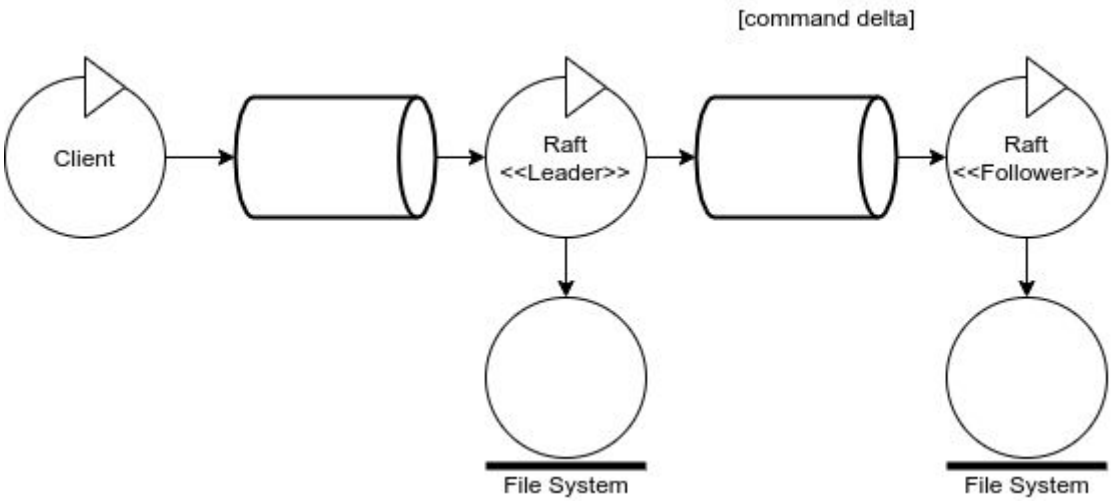
Este esquema de recupero admite la posibilidad de que se generen duplicados, retomar el procesamiento. El algoritmo tiene en cuenta esto, mediante el uso de una lista de procesados, compuesta por los ids de los elemento procesados entre el checkpoint y la falla.
En el caso particular que la falla se produzca al finalizar el stream de procesamiento, no es un problema ya que los acumuladores producen un único resultado por sesión y se puede utilizar también el identificador en la siguiente etapa para descartar el duplicado.

Para que el recupero pueda funcionar correctamente requerimos un almacenamiento estable. El mismo está implementado basado en el algoritmo Raft, que permite replicar comandos sobre máquinas virtuales distribuidas. En nuestro caso particular la máquina virtual es simplemente un diccionario en memoria y los comandos son las operaciones de agregar, sacar y recuperar datos.

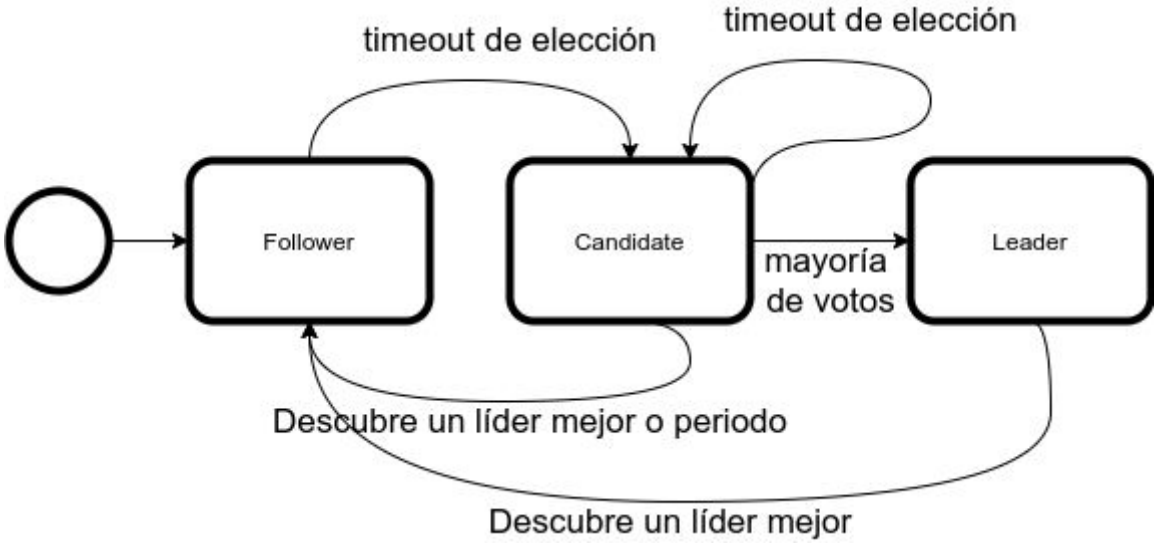
El algoritmo Raft tiene las siguiente características:

- Orientado a Leader y elecciones.
- Para un cluster de 2n+1 instancias tolera n fallas.
- Bien estudiado y utilizando en la industria.

Los clientes siempre interactúan con el líder del cluster, es el encargado también de mantener las réplicas al día. El líder dará por commiteado/replicados correctamente los elementos que puede confirmar están en la mayoría de las instancias. Y dichos elementos son los que pueden aplicarse a la máquina virtual. Es decir, solo los comandos que estén replicados mayoritariamente, será aplicado a la máquina virtual.
Esto quiere decir, que el cliente debe reintentar los comandos hasta tener un confirmacion del leader, y por otro lado es necesario que los comando sean idempotentes, afortunadamente, los comandos de la keyvalue store los son.



Frente a un fallo del líder la primera réplica en detectar la ausencia del líder, transiciona a un estado candidato, en el cual inicia un proceso de elección de líder.
Frente a una competición entre candidatos, ganará el candidato que tenga mejor historial de comandos, para Raft esto significa que el tenga el log con más elementos y el periodo mayor.



El siguiente diagrama presenta un escenario de elección de líder, donde 2 instancia compiten el voto de la tercera. Si el primero en llegar gana los votos, la competencia se resuelve por mayoría.

Monitorización:

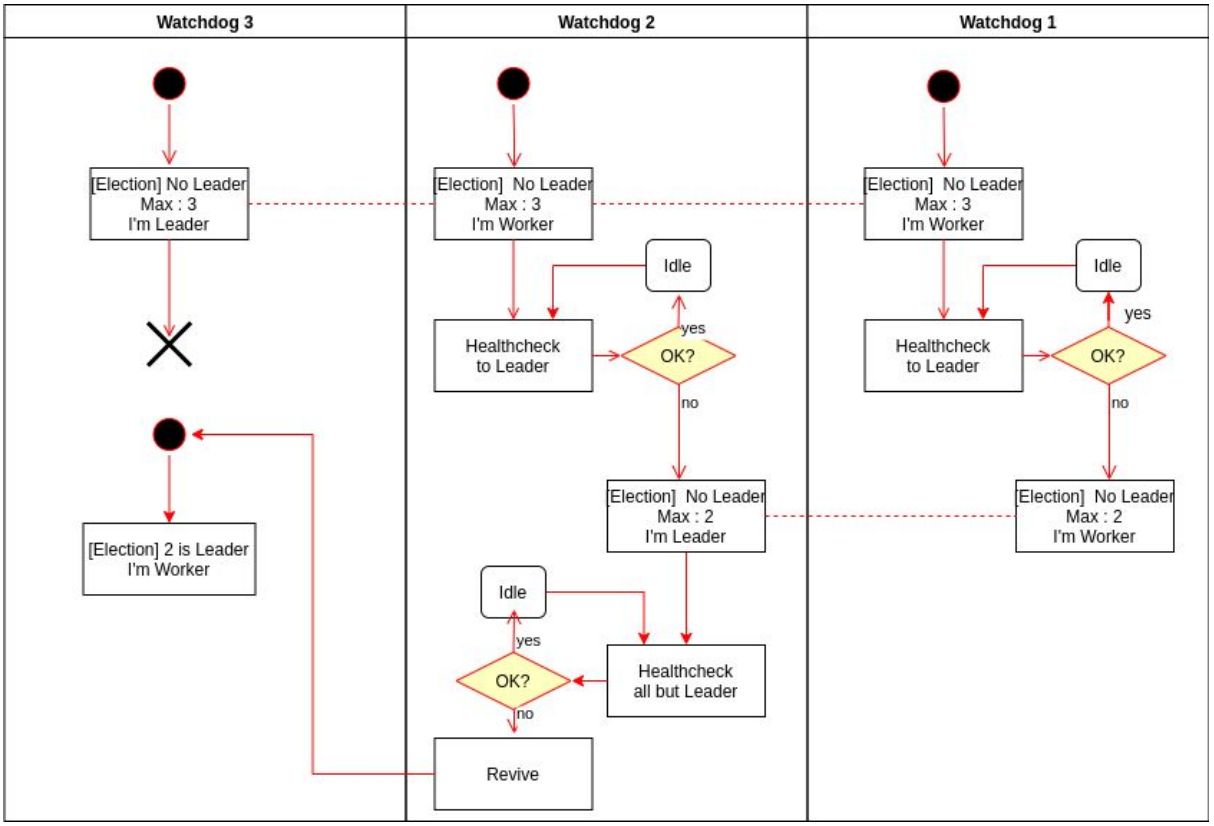
Si un nodo del sistema se cae necesitamos ser capaces de detectarlo y una vez hecho volver a levantarlo. Para ello implementamos un proceso “watchdog” el cual se encarga de realizar periódicamente request http del tipo “healthcheck” a cada nodo del sistema. En caso de recibir una respuesta KO o no recibir respuesta se encargará de disparar un restart del nodo en cuestión. Dentro de cada nodo del sistema corre un hilo encargado de atender los “healthcheck” requests.
Una limitación de la implementación actualmente es que el healthcheck es independiente del procesamiento, es decir este sistema de monitorización solo detecta si el nodo está caído pero no es capaz de garantizar que se encuentra verdaderamente operativo.
Como próximo paso para mejorar sistema de monitorización planteamos agregar al request de healthcheck una respuesta del estilo:

response = { isProcessing, lastProcessedTimestamp }

De modo tal que podamos verificar que si en cierto punto el nodo debería encontrarse activo, verdaderamente está procesando

Consenso entre réplicas:

Considerando la posibilidad de que los procesos encargados de monitorizar también pueden fallar, hemos de replicarlos, sin embargo solo deseamos que una de las réplicas ejecute el restart de un nodo caído. Para ello diferenciamos a una de las réplicas (leader) del resto (workers) consensuando mediante una elección quién es el leader. La elección consiste en consultar a todas las réplicas levantadas cual es leader y si ninguna lo es resuelve que el leader es la réplica levantada con mayor id. En el siguiente diagrama queremos mostrar el escenario en el cual falla el leader:



Ya que la elección, quizás es mejor pensarla como una decisión consensuada, se ejecuta de forma asincrónica e independiente en cada réplica. Para asegurar que el resultado de elección sea determinista hemos de considerar el siguiente supuesto:
El tiempo que tarda en levantar un nodo es mayor al tiempo que se tarda en detectar que está caído.
De manera tal que todas las réplicas tendrán tiempo de resolver la elección antes de que aparezca un nuevo candidato

Vista de Implementación:

Conceptualmente el procesamiento tiene la siguiente etapas:

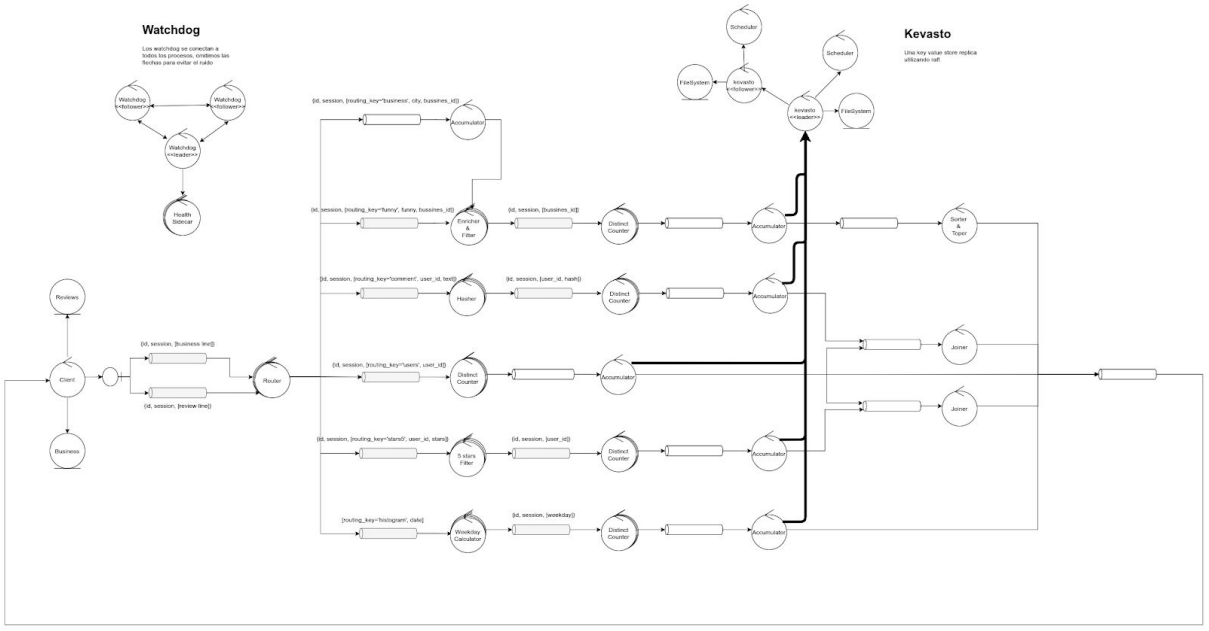
Input: Para simular el input de los datos se implementa un nodo cliente que lee los datos de un archivo y los publica en un exchange de un broker de mensajes en forma de chunks tal cual los lee.

Ruteo: El sistema toma estos mensajes desde los nodos router donde se filtra la información necesaria para realizar cada consulta y se publican distintivamente por tópicos. Para simplificar primero se envían/procesan los negocios y luego las reseñas ya que es necesario que se hayan procesado primero todos los negocios para poder procesar las reseñas en una de las consultas.

Mapeo: En esta etapa una serie de grupos de nodos realizan una función de mapeo, aquí uno de estos grupos en particular espera a recibir la información agregada de los negocios para empezar a procesar.

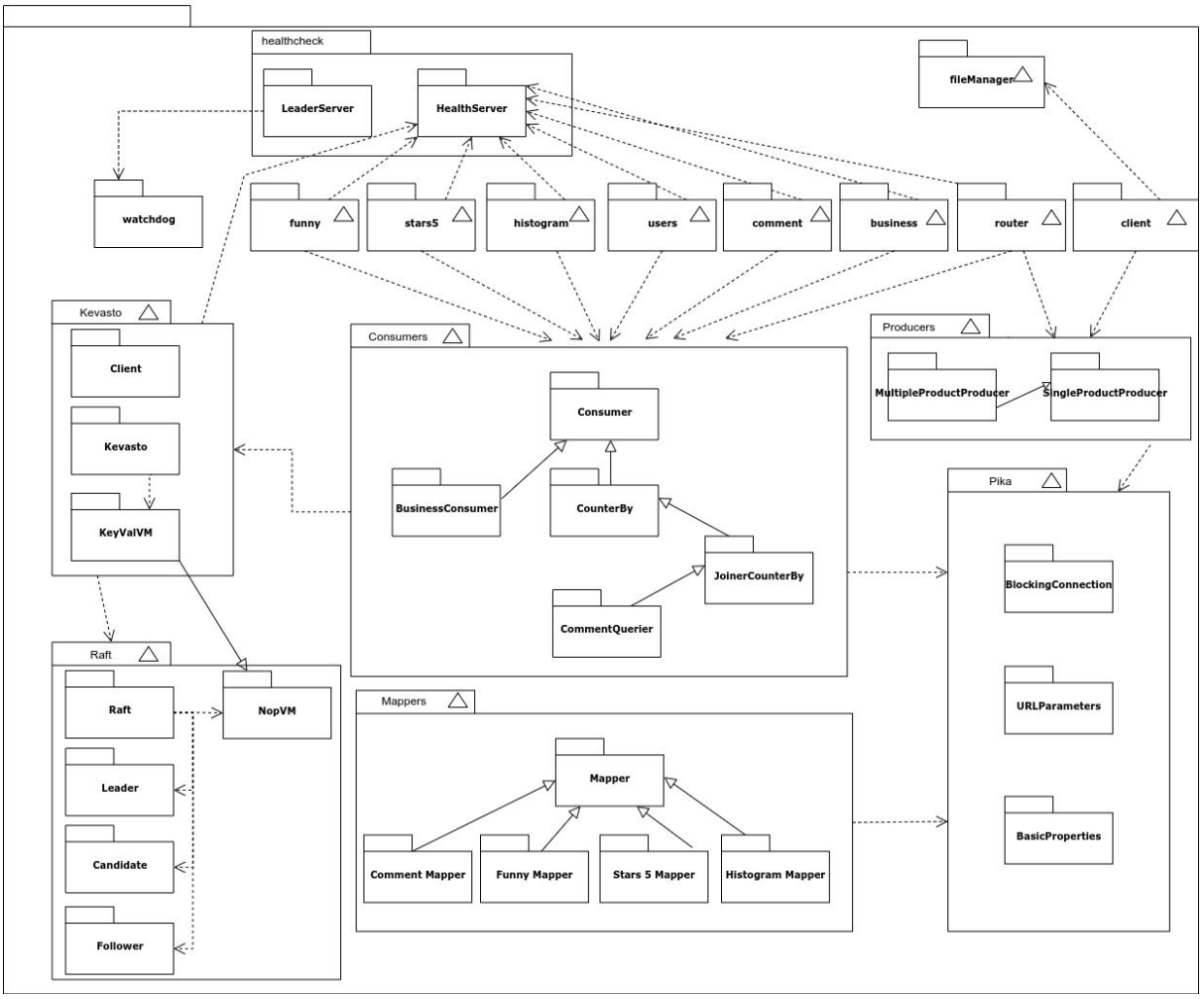
Agregación: A continuación viene la etapa donde esos datos se reducen según su criterio, en general se cuenta por clave. Finalmente, una vez con los resultados agregados, viene la etapa de junta, ordenamiento y filtrado según cada consulta. Una vez finalizada de procesar cada consulta se envía al cliente el resultado en una cola de respuesta cuya referencia va viajando por el sistema a medida que los componentes señalizan a la siguiente etapa el fin del procesamiento.

En este diagrama se muestran los distintos controladores del sistema y como se relacionan entre si. Las colas sombreadas representan que son nombradas, es decir que todos los controladores desencolan de la misma cola, mientras que las no sombreadas representan que son colas sin nombrar, por lo que cada controlador desencola una cola exclusiva. Los controladores desde la etapa de agregación en adelante se simplificaron en 1 solo por consulta, igual que se ve en el anterior diagrama.

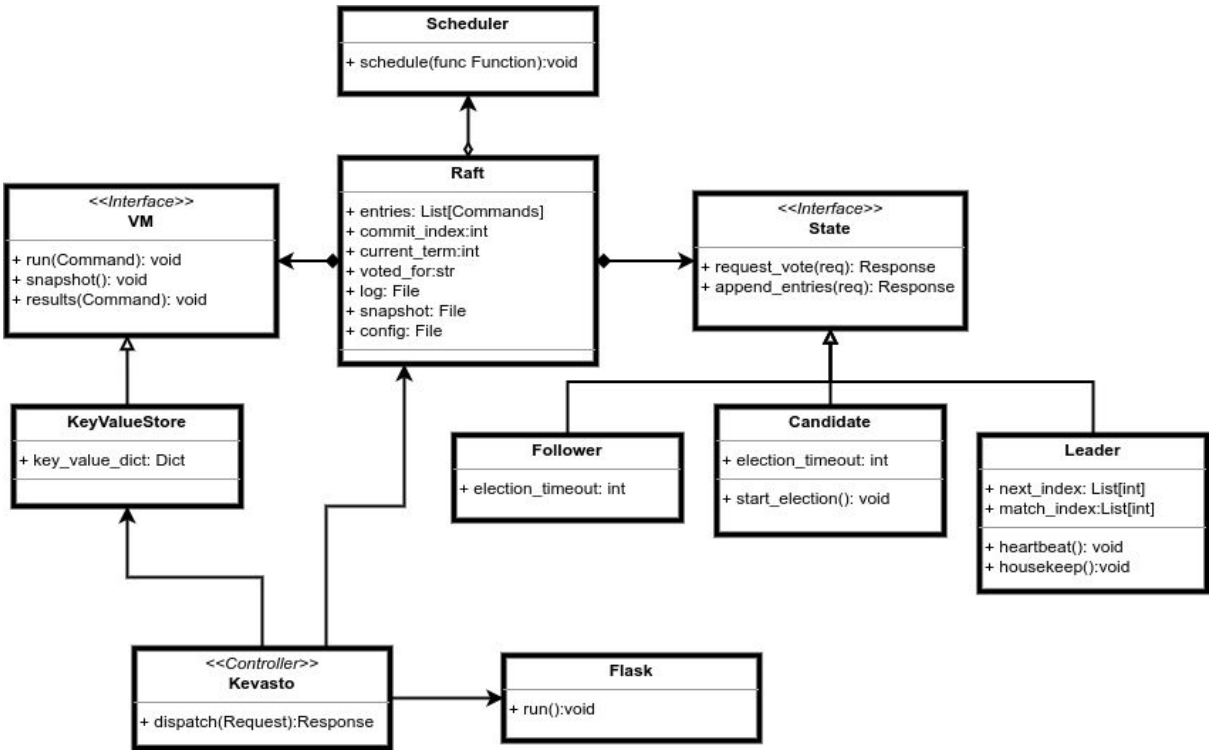


En el diagrama de paquetes hay 2 grupos principales que son los consumers y los mappers, que abstraen la lógica de agregación y mapeo respectivamente, a la vez tambíen abstraen la recepción/envío de mensajes. Quedó un poco mezclada la misma capa, eso debería estar dividido en 2 capas. Adicionalmente Tenemos los Paquetes correspondientes a la implementacion de Raft, a la Kevasto(Key Value Store) y los watchdog.

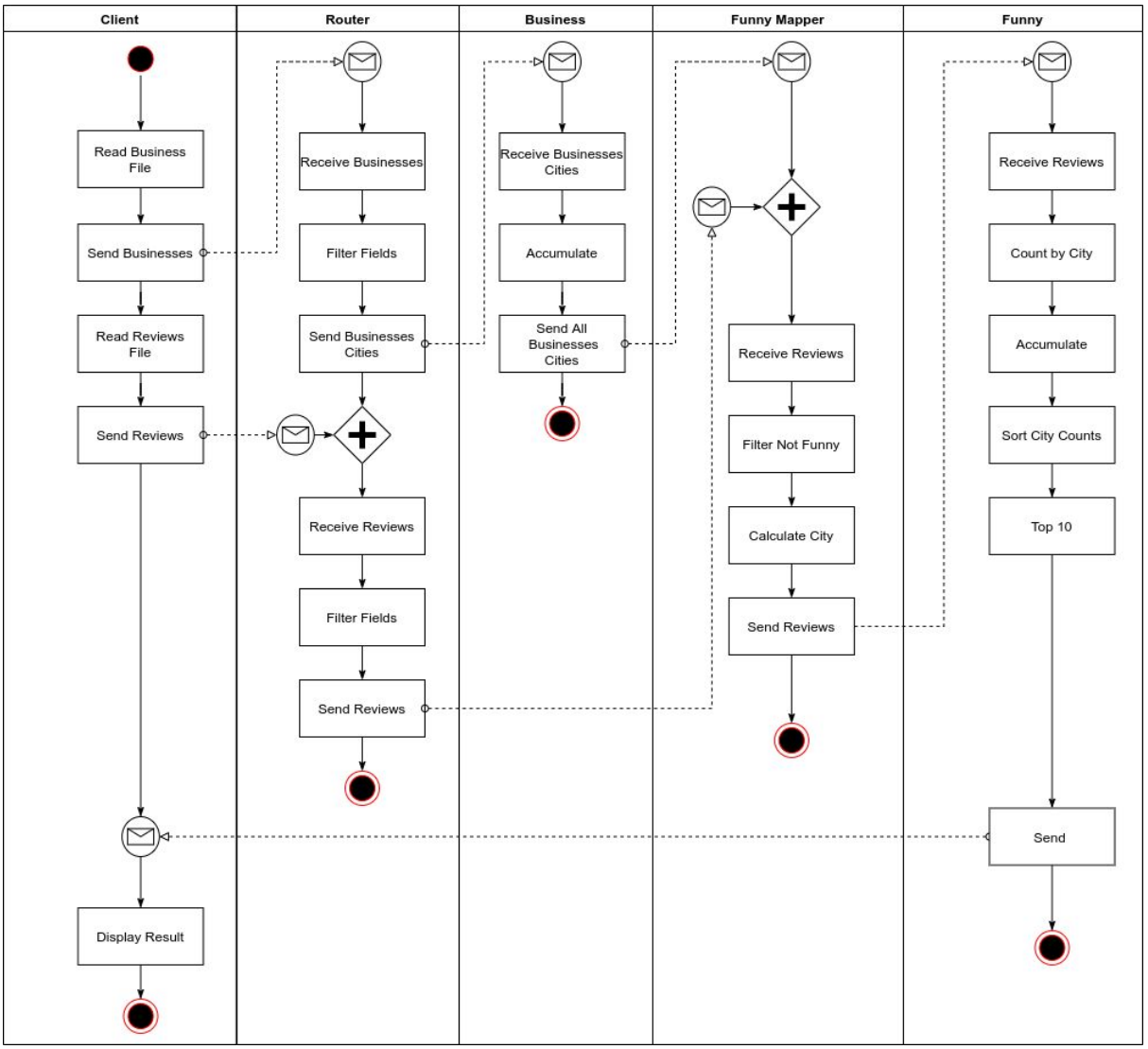
Yelp Reviews Analysis



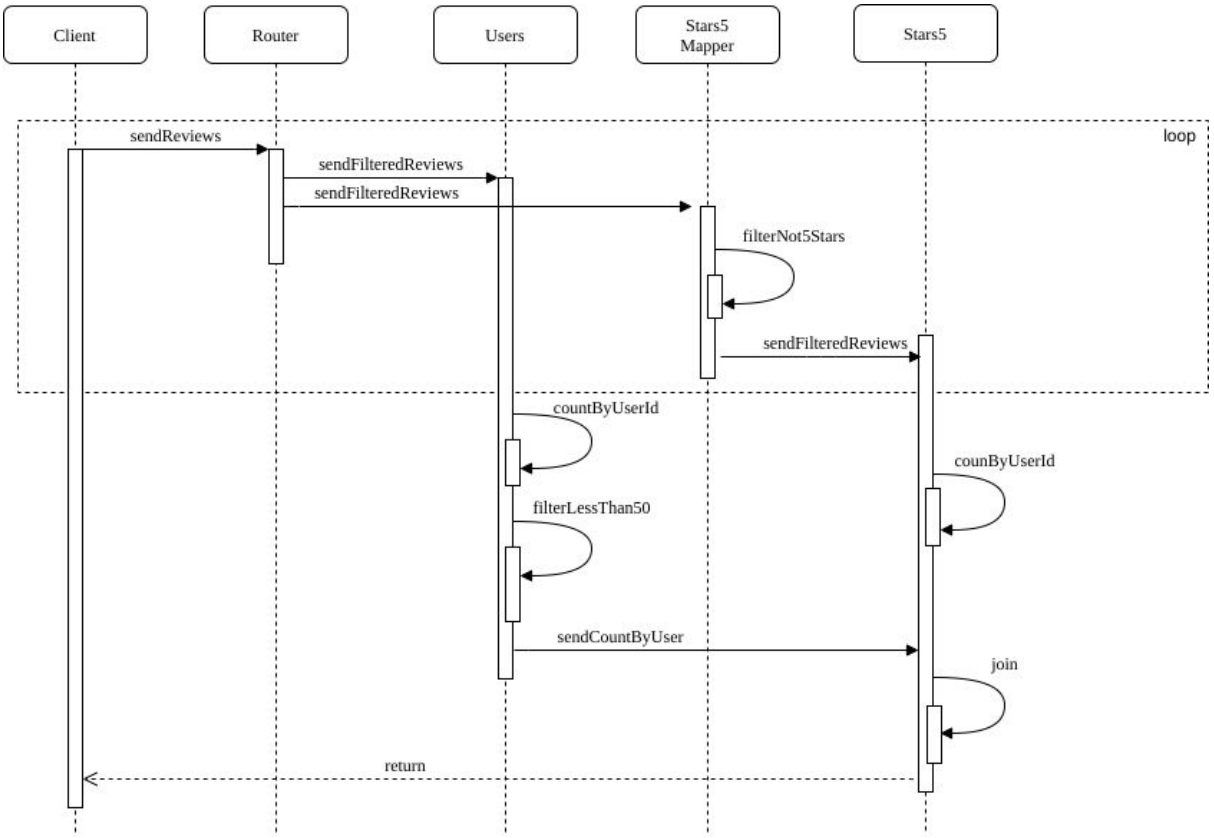
En particular entre el algoritmo de Raft y Kevasto tenemos la siguiente relación entre clases



Vista dinámica:

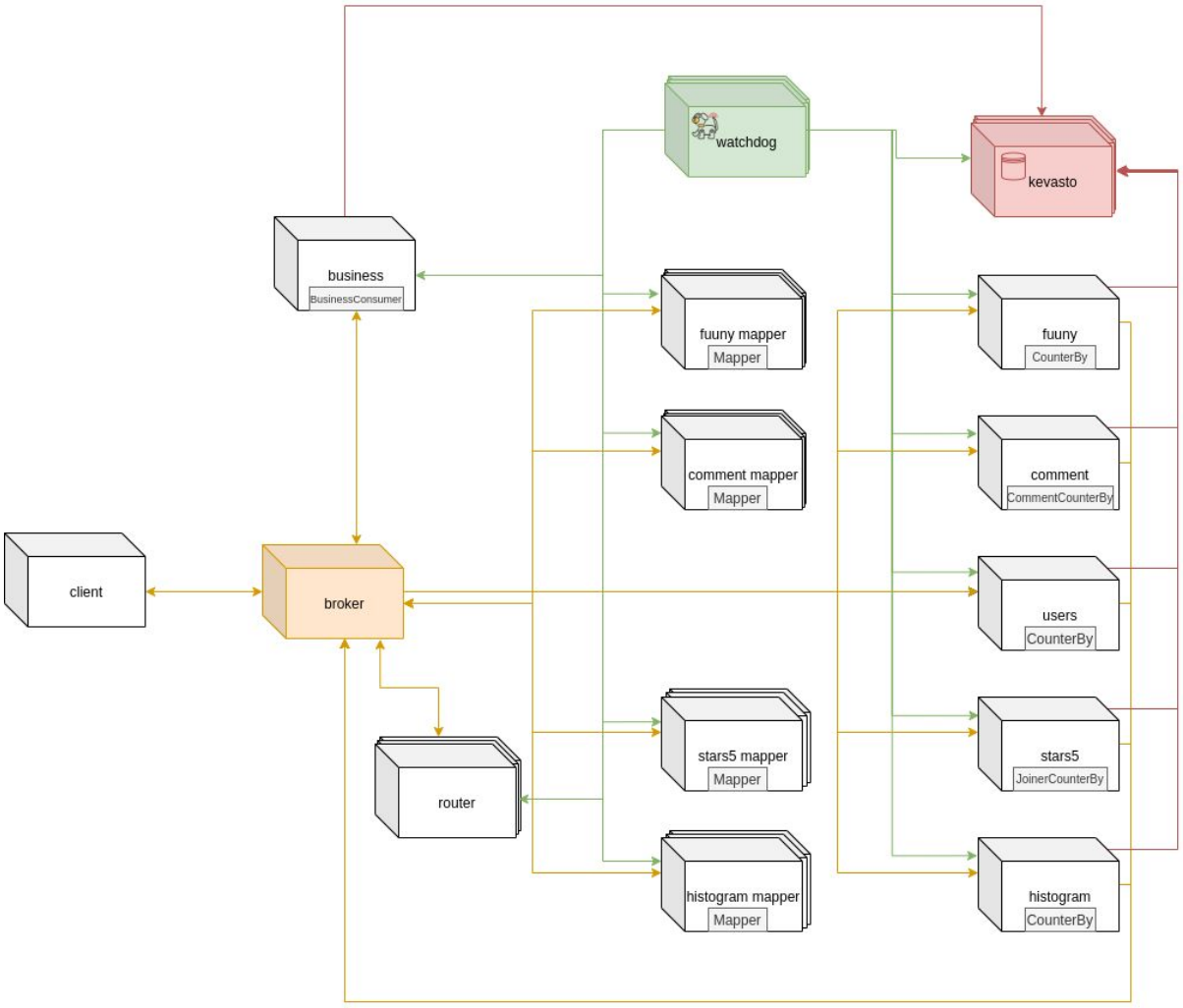


En este diagrama se intenta mostrar el flujo de la consulta de las 10 ciudades con reseñas de negocios divertidos. Se envían primero los negocios que son acumulados por el nodo Business y luego las reseñas que recién empiezan a ser procesadas cuando Funny Mapper tiene la información sobre la ciudad de cada negocio. Se filtran las reseñas que no fueron ingresadas como una experiencia divertida y las que si se les busca a qué ciudad corresponde. El nodo mapper cuenta las reseñas por ciudad y al recibir la señal de fin, ordena y responde al cliente con las primeras 10.



En este diagrama se intenta mostrar el flujo de la consulta de la consulta de usuarios con más de 50 reseñas todas con máxima puntuación. En un loop se envían todas la reseñas a los nodos Users y Star5 los cuales las cuentan por usuario, las que llegan a Star5 son filtradas en un nodo intermedio. Users filtra los usuarios con más de 50 reseñas y se envian en un único mensaje a Star5 quien se encarga de joinear por usuario y responder al cliente con los usuarios a los que no se le filtró ninguna reseña porque todas fueron puntuadas con la máxima puntuación.

Vista de despliegue:



Mostramos los nodos “físicos” presentes en la simulación, comunicándose entre sí con el broker como intermediario. Se puede ver también que toda la lógica de agregación, junta y lo que corresponda después de la agregación se agrupan en 1 nodo por consulta. A su vez todos los nodos están monitorizados por el watchdog mediante requests http.

Status:

La entrega actual contiene una implementación de los watchdogs completamente integrada y funcional. Es decir, frente a la falla de un proceso el watchdog los revive inclusive si se trata de el líder de los watchdogs.

La key value store también está completamente implementada y funcional, siguiendo lo mejor posible el pseudo código del paper de Raft, y pruebas preliminares indicarían que funciona correctamente. Pudimos comprobar que efectivamente el líder logra replicar su historial de eventos y que realiza snapshots cuando alcanza el máximo de archivo tolerado para el log de comandos. También están funcionales los casos de uso de poner, obtener de la kevasto.
Probando con la carga de trabajo requerida, hemos encontrado y corregido varias condiciones de carrera entre las votaciones y la transición de estado del algoritmo de raft, en esto lo mejor sería hacer bloqueantes/secuenciales todas las operaciones, en este momento solo ciertas operaciones están utilizando locks.

También, estamos soportando correr tandas consecutivas de trabajo de forma continua.

El mayor inconveniente que tenemos con la entrega es que no llegamos a modificar el algoritmo de procesamiento de los agregadores para funcionar correctamente, no tanto por la complejidad del algoritmo sino por una cuestión de tiempo. Creemos que esto se debe a un cambio que hicimos en el orden de consumo de las queue de rabbit que lleva a que los accumulators no avancen.

Otras mejoras tiene que ver con simplificar el código, la forma entre que propaga la queue de respuesta para producir el reporte utilizando properties de rabbit, se puede reemplazar, esto haría homogéneos los mappers y los consumers.