Cassandra: estudio a una base de datos NoSQL

J. Ccarhuas, O. Cetrà, G. Gumà, J. Velásquez

***Abstract*—En los últimos años, la demanda por bases de datos capaces de administrar cantidades enormes de datos se ha disparado. En un esfuerzo por solucionar este problema, ingenieros de Facebook crearon la base de datos Cassandra. Esta base de datos NoSQL, es actualmente usada por gigantes tecnológicos la cual les brinda un servicio de alta disponibilidad gracias a su arquitectura y a poder ser fácilmente configurable a cada situación debido a ser de código abierto. Cassandra tiene como objetivo ejecutarse sobre una infraestructura de cientos de nodos, los cuales se organizan en centros de datos ubicados en diferentes zonas geográficas y se replican los datos en diferentes nodos. A esta escala, los componentes fallan continuamente. La forma en que Cassandra gestiona el estado persistente frente a estas fallas impulsa la confiabilidad y escalabilidad de los sistemas de software que dependen de este servicio sin tener un único punto de fallo. El sistema Cassandra fue diseñado para ejecutarse en hardware básico económico y manejar un alto rendimiento de escritura sin sacrificar la eficiencia de lectura.**

I. Introducción

Cassandra es un sistema distribuido descentralizado de base de datos NoSQL de código abierto, que sirve para manejar un gran volumen entre diversos servidores ubicados en varias zonas geográficas, lo que se conoce como "Big Data". Este sistema se caracteriza por ofrecer una alta disponibilidad, escalabilidad y tolerancia a fallos. Usa el lenguaje CQL (Cassandra Query Language), único de Cassandra. Su sistema de administración de base de datos se orienta en columnas y su paradigma de almacenamiento es mediante clave-valor. La aplicación de Cassandra está escrita en Java.

Cassandra fue desarrollada para potenciar la búsqueda en la bandeja de entrada de Facebook por Avinash Lakshman y Prashant Malik. Está inspirado en el diseño de distribución Amazon Dynamo de 2007 y en el modelo de datos de Google BigTable de 2006. Fue publicado en 2008 por primera vez, en 2009 se convirtió en proyecto open source por Apache Software Foundation Incubator y en 2010 pasó a ser un proyecto top-level de la fundación Apache. Actualmente la compañía que lo mantiene y distribuye es DataStax.

Cassandra fue diseñada para dar solución a casos como Facebook, una red social que atiende cientos de millones de usuarios utilizando decenas de miles de servidores en horas pico[1]. Dado que los usuarios son atendidos desde data centers que están distribuidos geográficamente, poder replicar datos en todos los centros es clave para mantener bajas las latencias de búsqueda. Uno de los principales problemas para este tipo de plataformas es su gran crecimiento exponencial. Para mantener unos estándares de rendimiento, confiabilidad y eficiencia, esta ha de ser altamente escalable. Otra problemática a resolver en una infraestructura compuesta por miles de componentes es enfrentar fallos; siempre hay un número pequeño pero significativo de componentes de red y/o servidores que fallan en un momento dado, por lo que el diseño software debe construirse de una manera que trate los fallos como una forma normal de funcionamiento y no como una excepción. Cassandra ha cumplido con su funcionamiento de forma impecable y ahora está implementado como el sistema de almacenamiento backend para múltiples servicios dentro de Facebook. Ha trascendido tanto que incluso se ha convertido en la base de datos de grandes empresas como Netflix, Twitter, Spotify, Ebay y Uber, entre otros.

En este artículo, se abordará los principales factores por el cual Cassandra se ha convertido en el pilar de la base de datos de grandes empresas y una de las más importantes de tipo NoSQL. En la sección 2 se introduce el teorema CAP y se especifica que dos características prioriza Cassandra en su implementación. En la sección 3 se habla del modelo de datos; cómo se estructuran las filas y columnas dentro de un mismo keyspace. En el apartado 4 se explica la arquitectura y las estrategias que se emplean en este sistema distribuido de bases de datos para abordar las diferentes problemáticas. En la sección 5 se explican los casos de uso de Twitter y Netflix. En el apartado 6 se hace una comparativa con Cassandra, MongoDB y HBase. Por último, en la sección 7 se hace una breve conclusión del estudio.

II. Estado del Arte

En la actualidad Cassandra está proclamada como una de las primeras opciones de las bases de datos NoSQL cuando se trabaja con Big data. En el ranking db engines [2] se posiciona en el décimo puesto. Cabe recordar que el uso de Cassandra solo es rentable cuando se trabaja con una gran cantidad de datos, ya que los recursos que se necesitan, a pesar de ser inferiores respecto otras soluciones alternativas, siguen siendo mayores que la que pudiese requerir un SQLite o MySQL, por eso que Cassandra esté tan arriba en este ranking da mucho qué decir sobre su potencial. Entre las NoSQL se encuentra en el tercer puesto por detrás de MongoDB [3], pero este ranking es de tendencia de los usuarios a utilizarla. Si comparamos la dificultad de ambas tecnologías, MongoDB es mucho más fácil de utilizar y no requiere de tantos recursos como Cassandra.

Hasta ahora se ha dicho que Cassandra requiere muchos recursos y además de que su aprendizaje requiere más tiempo que otras tecnologías. Por eso DataStax, que es la empresa que mantiene la versión que se distribuye de Cassandra, en el ámbito comercial, tiene como objetivos hacer más fácil el uso al usuario y poderla ejecutar de forma particular con recursos mínimos. Actualmente hay una versión beta de Apacha Cassandra, la cual es la 4.0 [4]. Esta versión manifiesta de que Cassandra se quiere adaptar a la segunda ola de datos, causada por el 5G, mejorando por 5 su velocidad en operaciones de escalabilidad como la incorporación de nuevo nodos. Introduce una herramienta propia de monitoreo, que

además permite gestionar la seguridad del sistema. Soporta una versión más nueva de Java que es la 11. Es la versión beta aún, por lo que se tendrá que ver como esta nueva versión se adapta a las necesidades actuales.

III. Teorema CAP

Cassandra es uno de los sistemas de almacenamiento de datos de tipo NoSQL, el cual significa Not Only SQL. Estos tipos de bases de datos están optimizados específicamente para aplicaciones que requieren grandes volúmenes de datos, baja latencia y modelos de datos flexibles, lo que se logra mediante la flexibilización de algunas de las restricciones de coherencia de datos en las bases de datos relacionales[5]. NoSQL se utiliza para describir una clase de bases de datos no relacionales que escalan horizontalmente.

Sin embargo, el teorema CAP se ha convertido en un modelo útil para describir el comportamiento fundamental de los sistemas NoSQL. El teorema CAP establece que es imposible que un servicio distribuido sea consistente, disponible y tolerante a la partición simultáneamente. Definimos estos términos de la siguiente manera.

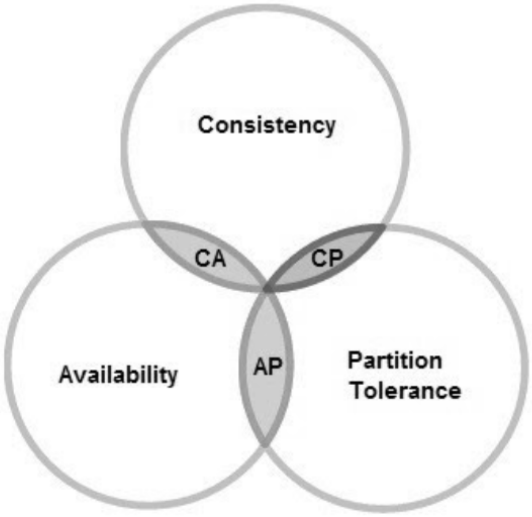

Fig. 1. Teorema CAP [25]

Consistencia: Todos los nodos deben garantizar la misma información en el mismo instante de tiempo cuando insertamos datos, al actualizar datos y si consultamos datos. Para tal efecto la comunicación entre los nodos debe ser de suma importancia.

Disponibilidad: Independientemente si uno de los nodos se ha caído o ha dejado de emitir respuestas, el sistema debe seguir en funcionamiento y aceptar peticiones tanto de escritura como de lectura. Una vez se pierde comunicación con un nodo, el sistema automáticamente debe tener la capacidad de seguir operando mientras este se restablece y una vez lo hace, se debe sincronizar con los demás.

Tolerancia al particionamiento de red: El sistema debe estar disponible aunque existan problemas de comunicación entre los nodos, cortes de red que dificultan su comunicación o cualquier otro aspecto que genere su particionamiento. Por lo general los ambientes distribuidos están divididos geográficamente, donde es normal que existan cortes de comunicación entre algunos nodos, por lo cual, el sistema debe permitir seguir funcionando aunque existan fallas que dividan el sistema.

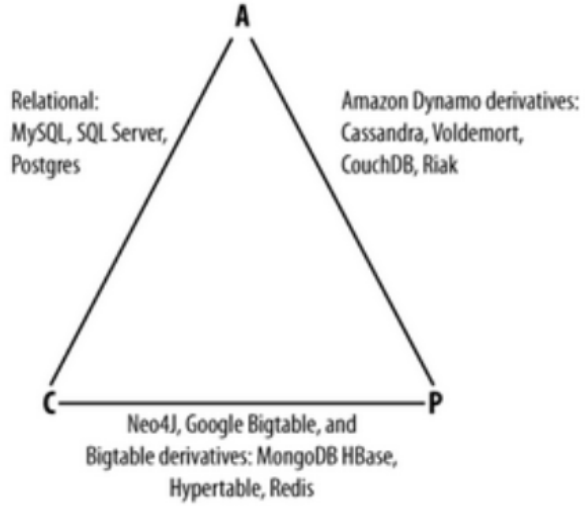

Fig. 2. Bases de datos categorizados en el teorema CAP [6]

Cassandra se caracteriza por ser un sistema AP, proporcionando alta disponibilidad y tolerancia al particionado. Es decir, el sistema siempre estará disponible a las peticiones aunque se pierda la comunicación entre algunos nodos[6]. El nivel de consistencia en Cassandra es configurable y totalmente dependiente del transcurso del tiempo, cómo veremos en el apartado de niveles de consistencia dentro de la arquitectura. Es por esta razón que se le otorga un nivel de consistencia eventual. Por lo tanto, en proyectos donde las consultas se van a hacer bastante más tarde que las escrituras, tendremos máxima consistencia. Se puede configurar Cassandra con un nivel de consistencia alto para poder proporcionar consistencia en lugar de disponibilidad, ofreciendo un servicio CP en el teorema CAP, aún que no es su configuración más estándar.

IV. Modelo de datos

Cassandra se basa en un modelo híbrido entre un modelo clave-valor y orientado por columnas como BigTable[7][8]. A diferencia de un RDBMS (Relational Database Management System) típico que consta de filas, tablas y columnas, Cassandra consta de columnas, super columnas, filas, familias de columnas, super familias de columnas y keyspaces. A continuación vamos a ver las características de estos diferentes niveles dentro de la base de datos.

Columna: es el elemento básico del sistema, es una estructura de tres campos que contienen el nombre, el valor y una marca de tiempo.

Nombre: es el identificador con el que podremos acceder a ella para obtener o modificar el valor que contiene. Es único y no puede haber dos iguales en el mismo conjunto de columnas.

Valor: es el dato de una columna. Es el único elemento modificable por el usuario.

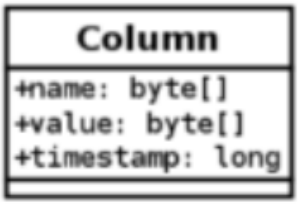

Fig. 3. Valores del elemento columna [7]

Marca de tiempo: nos indica cuándo se modificó por última vez esa columna. Este elemento se genera automáticamente al cambiar el campo valor. Se utiliza para la resolución de conflictos y así diferenciar cuando se introdujo un dato u otro[9].

Super columna: las super columnas se agrupan como columnas con un nombre común y son útiles para modelar tipos de datos complejos.

Fila: conjunto de columnas identificado por una clave única para acceder a él. Las columnas que componen una fila son únicamente las que tienen un valor establecido, a diferencia de las bases de datos relacionales que mantienen la columna asignándole un valor null en ella.

Fig. 4. Filas del modelo de la base de datos [7]

Familia de columnas: es una agrupación de un conjunto de filas. Las familias de columnas representa cómo están estructurados los datos. También se pueden agrupar las familias de columnas en super familias de columnas.

Keyspace: es la unidad de nivel superior de información en Cassandra. Es donde se almacenan todos los datos, generalmente, de una aplicación[10].

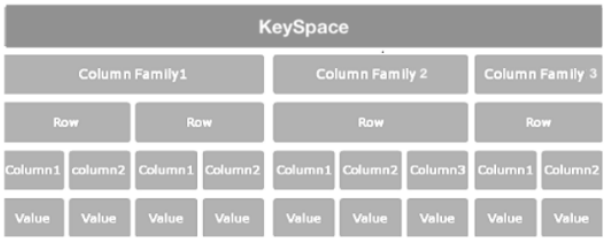

Fig. 5. Modelo completo Base de Datos [8]

A. Clave

La clave primaria es la encargada de ser el identificador único de un conjunto de columnas que se agrupan como una única fila. Cassandra usa una función hash consistente para distribuir los datos en los diferentes nodos[11]. Hay distintos modelos de claves[12]:

Clave primaria o clave de partición: es el método más simple, donde se aplica una función hash para determinar un número de clave primaria al cual estara asociado la fila. La salida de la función se utiliza para determinar qué nodo, y réplicas, obtendrán los datos. El algoritmo utilizado es Murmur3[13].

Clave primaria compuesta: esta clave consta de dos partes: la clave de la partición y la clave de la agrupación. La clave de partición es el mismo concepto que en el primer caso, mientras que el trabajo de la clave de agrupación es agrupar y organizar datos de una partición para permitir la realización de consultas eficientes mediante las columnas de la fila.

V. Arquitectura

La base de datos de Cassandra está diseñada para distribuirse entre varias máquinas que operan juntas y que aparecen como una sola instancia para el usuario final. Entonces, la estructura más externa en Cassandra es el cluster. Los nodos interactúan entre sí en un modelo descentralizado mediante una estructura de mensajes peer-to-peer, donde no existe jerarquía alguna y todos actúan por igual dentro del grupo. Dado que Cassandra es distribuida y descentralizada, no existe un único punto de fallo y ofrece una alta disponibilidad. Los datos se asignan a los nodos del cluster organizándose en un anillo, donde se originan diferentes rangos para determinar qué nodos deben replicar información de otros.

Cassandra se utiliza con frecuencia en sistemas que abarcan ubicaciones físicamente separadas. Cassandra proporciona dos niveles de agrupación que se utilizan para describir la topología de un clúster: centro de datos y rack. Un rack es un conjunto lógico de nodos muy próximos entre sí. Un centro de datos es un conjunto lógico de racks, ubicados en el mismo edificio y conectado por una red confiable.

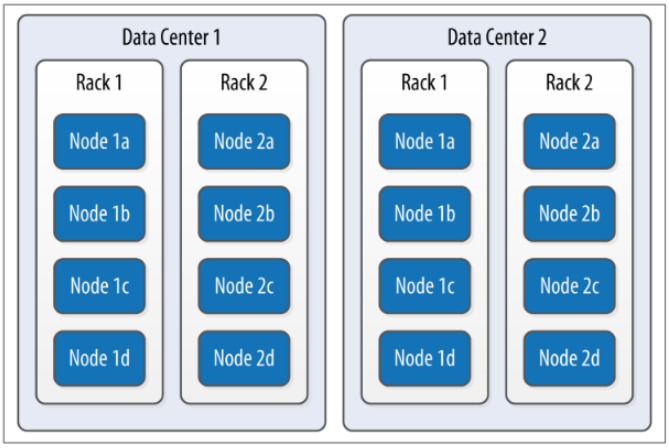

Fig. 6. Topología de un clúster con centros de datos, racks y nodos [6]

Cassandra almacena copias de sus datos en varios centros de datos para maximizar la disponibilidad y la tolerancia a la partición, mientras que prefiere enrutar las consultas a los nodos del centro de datos local para maximizar el rendimiento.

A. Gossip y detección de fallos

Para poder soportar la descentralización y la gran tolerancia al particionamiento que ofrece Cassandra, se utiliza el protocolo Gossip, que permite la comunicación entre los nodos de un mismo cluster. Este protocolo se implementa con la principal finalidad de detectar fallos en la red o nodos caídos, por lo que se asume que la red es defectuosa. Por eso este protocolo también se encarga de disparar el proceso de Hinted Handoff que se encarga de reducir el tiempo necesario para que un nodo que falla temporalmente vuelva a ser coherente con los activos[14]. El funcionamiento del protocolo es el siguiente: Periódicamente un nodo aleatorio que éste entre los activos será el Gossiper, este escogerá un nodo aleatorio con el cual empezar una gossip session. Cada ronda de gossip requiere de 3 mensajes. El gossiper envia al nodo seleccionado un mensaje llamado **GossipDigestSynMessage**. Cuando el nodo recibe el mensaje, le devuelve un **GossipDigestAckMessage**. Finalmente cuando el Gossiper recibe el ack, envía al nodo seleccionado un **GossipDigestAck2Message**, para acabar la ronda Gossip. Si el gossiper determina que un nodo está muerto, lo marca en una lista que tiene en local y lo logea a otros nodos.

Para la detección de fallos Cassandra utiliza el algoritmo llamado *Phi Accrual Failure Detection*. Este algoritmo sigue dos ideas principales, la primera es la flexibilidad, por lo que tiene que estar desacoplado de la aplicación. La segunda es la sospecha, en vez de declarar un nodo caído o activo según la latencia de conexión, este algoritmo establece un nivel de sospecha por lo que se barajan diferentes opciones como la que un nodo no esté disponible por un problema temporal en la red o por fallas del sistema que pueden ser puntuales. Este detector de fallos de Cassandra permite las siguientes operaciones:

isAlive: que reporta si el nodo está activo o no.

interpret: utilizado por el Gossiper para decidir si el nodo está activo o no según el nivel de sospecha que ha calculado el algoritmo.

report: cuando un nodo recibe un isAlive, invoca automáticamente este método.

B. Snitches

El funcionamiento de los snitches es mejorar el tiempo de conexión entre nodos. Para eso reúne la información sobre la topología de la red, de manera que Cassandra puede establecer rutas entre los nodos según su localidad geográfica y proximidad. El snitch averigua dónde están los nodos y la relación con otros, y si la información de estos está replicada en otros. Los snitches basan su funcionamiento en la comparación de los octetos de las direcciones IP de los nodos. Si dos nodos tienen el mismo valor en el segundo octecto, estos están determinados en el mismo data center. Si tienen el mismo valor en el tercer octeto están en el mismo rack. Todas las afirmaciones anteriores son la que Cassandra ha sacado basándose en la suposición de cómo están sus servidores ubicados en diferentes subredes.

C. Tokens en el anillo

A un clúster de Cassandra se le llama normalmente anillo por su disposición de arquitectura peer-to-peer. A cada nodo en este anillo se le asigna un token único y cada uno de estos nodos reclama la posesión de un rango de valores o llaves entre su token y el token del siguiente nodo. Esto está representado en la clase org.apache.cassandra.dht.Range. La tarea de cómo se representa cada nodo recae en los particionadores.

D. Particionamiento

El propósito de un particionador es permitirte especificar como las llaves de las columnas están ordenadas, cosa que tiene un impacto importante en cómo los datos van a estar distribuidos por los nodos. Además tiene efecto en las opciones disponibles para búsquedas en un rango de filas. Hay diferentes particionadores que se explican a continuación.

Particionador aleatorio: Este particionador es implementado por org.apache.cassandra.dht.RandomPartitioner y es el usado por defecto en Cassandra. Usa un BigIntegerToken con un hash MD5 que determina dónde estarán puestas las diferentes llaves en el anillo de nodos. Su principal ventaja es la distribución en partes iguales en todo el clúster al ser totalmente aleatoria. El principal inconveniente es causar rangos de búsqueda ineficientes, ya que estas mismas llaves pueden estar en lugares muy dispares y los datos devueltos estarán en un orden aleatorio.

Particionador conservador de orden: Este particionador es implementado por org.apache.cassandra.dht.OrderPreservingPartitioner. Usando este particionador, el token es un UTF-8 string. Las columnas estarán guardadas por orden de su llave alineada igual a la estructura física. Es importante saber que este tipo de particionamiento no es más eficiente que el particionamiento aleatorio, tan solo provee un mejor orden. Tiene el inconveniente de crear un anillo que puede ser muy desequilibrado. Con este tipo de particionamiento es muy probable que se pueda acabar con muchos datos en las primeras posiciones de la BD y muy pocos en las últimas. Al usar este particionamiento, es posible que sea necesario rebalancear nodos periódicamente usando herramientas como el loadbalance o move de Nodetool.

Particionador conservador de orden comparador: Este particionador ordena las llaves según la configuración regional del inglés de los Estados Unidos. Al igual que el anterior particionador, requiere que las llaves sean UTF-8 string. Este particionador es usado raramente, ya que su utilidad es muy limitada.

Particionador ordenador por bytes: Este particionador, nuevo de la versión 0.7, preserva el orden tratando todos los datos como bytes sin tener que convertirlos en strings. Es el más óptimo cuando es necesario no tener que validar las llaves como strings.

E. Estrategias de replicación

La alta disponibilidad del sistema se logra mediante la replicación de datos en diferentes máquinas y diferentes centros

de datos. La replicación de datos es asíncrona, en la que las escrituras de réplicas se realizan en segundo plano. Esto permite tiempos de respuesta más rápidos, aunque con una menor garantía de coherencia de los datos[15].

Para un grupo de N nodos, el factor de replicación (RF) es la cantidad de veces que una clave se replica en el grupo. Cada nodo coordinador está a cargo de almacenar sus datos localmente y replicarlos en los nodos RF-1. Por lo tanto, un nodo almacenará RF/N de los rangos de claves[16].

Cada nodo de Cassandra tiene una réplica de algo. Al crear una réplica, la primera siempre se coloca en el nodo que reclama la clave del rango de su token. Todas las réplicas restantes se distribuyen de acuerdo con una estrategia de replicación, que veremos ahora. La estrategia de réplicas se vuelve más importante cuantos más nodos agreguen en el clúster.

Estrategia de Rack-Unware: es la estrategia más simple y la que se usa predeterminadamente. Esta estrategia coloca réplicas en un solo centro de datos, de una manera que no es consciente de su ubicación en un rack de un centro de datos. Esto significa que la implementación es teóricamente rápida, pero no si el próximo nodo que tiene la clave está en un rack diferente al de los demás. Esta estrategia no tiene en cuenta los centros de datos, por lo que las réplicas solo son en su mismo centro.

Estrategia Rack-Aware: también conocida como "estrategia vieja de topología de red". Se utiliza principalmente para distribuir datos en diferentes racks. Supongamos que tiene dos centros de datos, DC1 y DC2, y un conjunto de servidores. Esta estrategia colocará algunas réplicas en DC1, colocando cada una en un rack diferente y colocará otra réplica en DC2. La estrategia Rack-Aware se usa específicamente para cuando tiene nodos en el mismo clúster de Cassandra distribuidos en dos centros de datos y se está utilizando un factor de replicación de 3.

Estrategia de topología de la red: permite especificar cómo se deben colocar las réplicas en los centros de datos que la estrategia anterior. A diferencia de la Rack-Aware, está pensada para replicar en más de un nodo de otro data center la información.

F. Niveles de consistencia

El nivel de consistencia es el número de réplicas en las que la operación de lectura/escritura deben ser validadas antes de devolver un mensaje de éxito al cliente. Un nivel de consistencia alto significa que varios nodos necesitan responder a la consulta, brindándole más seguridad al cliente y al sistema de que los valores presentes en cada réplica son los mismos.

Si dos nodos responden con una marca de tiempo diferente, la mas reciente sera la que se envíe al cliente. En segundo plano, Cassandra ara un "reparo de lectura": actualizará el valor de las réplicas con la información que contiene la columna de la marca de tiempo más reciente.

1) Niveles de consistencia de lectura:

Uno: inmediatamente se envía al cliente el valor del primer nodo que responda. En segundo plano se crea un thread para verificar que otras réplicas tengan el mismo valor y se actualiza en caso de no ser así.

Quorum: cuando la mayoría de las réplicas ((factor de replicación/2)+1) han respondido, se envía al cliente el valor con la marca de tiempo más reciente. Si es necesario se hace un reparo de lectura.

Todos: se espera a la respuesta de todos los nodos y se envía el valor más reciente. Se realiza un reparo de lectura si es necesario.

2) Niveles de consistencia de escritura:

Cero: se acepta la petición sin enviar un mensaje de respuesta al cliente.

Cualquiera: se devuelve un mensaje de éxito si se ha realizado la escritura en alumnos un nodo.

Uno: el valor se escribe en el log y la memtable al menos en un nodo.

Quorum: la escritura es exitosa cuando la mayoría de las réplicas ((factor de replicación/2)+1) han recibido el valor.

Todos: antes de retornar un éxito al cliente se debe haber recibido en todas las réplicas.

G. Caching

En Cassandra se permiten distintas estrategias y configuraciones referentes al almacenamiento de la caché, que se determinarán en función de las necesidades de nuestro sistema. Existen dos caches principales integrados al Cassandra: un caché de filas y otro para las claves. En el primero se almacenan aquellas filas que estén completas por lo que es un superconjunto del caché de claves. Es decir, si se usa un caché de filas no será necesario el uso del de claves.

La estrategia de almacenamiento se ajustará en función de tres factores: las consultas que realicemos, la relación entre el tamaño de la pila con la de la caché y, finalmente, la relación de tamaño entre las filas y el tamaño de sus llaves.

H. Staged Event-Driven Architecture (SEDA)

Cassandra implementa una arquitectura basada en eventos por etapas (SEDA Staged Event-Driven Architecture). SEDA es una arquitectura general usada para servicios de Internet concurrentes diseñada en 2001. En el caso de Cassandra una operación la empieza a ejecutar un thread, que este, delega en otro thread y este podría en otro, y así sucesivamente. Este trabajo se divide en diferentes etapas(stage). Una etapa es la unidad mínima de trabajo que será ejecutada por los distintos threads. Aparte de la operación a ejecutar, cada etapa contiene también el apuntador a la siguiente etapa a ejecutar.

Este diseño de SEDA permite a Cassandra optimizar el uso de sus recursos, ya que las distintas etapas se irán ejecutando en función de los recursos disponibles en este momento.

I. Escalabilidad y Elasticidad

Cassandra ofrece una alta escalabilidad por su arquitectura. Se caracteriza por proveer una escalabilidad horizontal, donde al agregar un nodo al sistema este le asigna unos tokens y replica la información que le corresponde, cómo un nodo

más. Esto permite aceptar más peticiones, puesto que al agregar un nodo este puede aceptar consultas de los datos que dispone. El software en sí tiene un mecanismo interno para mantener sus datos sincronizados con los demás nodos del clúster, cómo se ha explicado anteriormente en el apartado de replicación. Específicamente, esta escalabilidad es elástica. Esto significa que el clúster puede escalar o desescalar sin problemas. Reducir, por supuesto, significa eliminar parte de la capacidad de procesamiento del anillo. Esto sucede cuando se mueve partes de la aplicación a otra plataforma o si la aplicación pierde usuarios, no requiere de tanta capacidad de cómputo o los clientes de la aplicación han bajado.

La escalabilidad ofrece un rendimiento que aumenta linealmente, lo que quiere decir que el rendimiento de forma lineal respecto al número de nodos que añadamos. Por ejemplo, si con 2 nodos soportamos 100.000 operaciones por segundo, con 4 nodos soportaremos 200.000.

Cassandra no requiere de recursos muy específicos, para un servidor de producción mínimo requiere de tan solo 2 cores y 8GB de RAM, para un servidor de producción decente las grandes empresas utilizan 8 cores y 32 GB de RAM[17]. Tan solo con esto Cassandra ya es altamente concurrente, soportando múltiples peticiones (tanto de lectura o escritura), utilizando múltiples threads ejecutándose en múltiples núcleos. Cassandra utiliza mucha RAM para la compresión de la metadata, filtros, proceso de filas, llaves y caches, por eso para su implementación requiere de instancias preparadas. Para su implementación a nivel comercial muchas empresas utilizan diversos clouds, a través de servicios como los de Azure o AWS, este último dispone de instancias específicas para Cassandra. Respecto a sus sistemas de ficheros, Cassandra se diseñó con la idea de tener información redundante, por si algún nodo cae, por eso se debe evitar utilizar NFS o SAN, se recomienda el uso de RAID0 o JBOD. Todo esto permite a Cassandra tener una escalabilidad horizontal, si la arquitectura está bien planteada desde el principio.

VI. Casos de uso

A. Twitter

Los almacenes de datos (Data WareHouses) que hasta hace no mucho trabajaban con RDBMS, comenzaron a mostrar sus limitaciones cuando los datos comenzaron a crecer exponencialmente, debido al ”boom” del Big data, aún más cuando se trata de gestión en “tiempo real”. Por eso las bases de datos NoSQL que se han desarrollado con el propósito de soportar aplicaciones Big data han cogido el relevo. Twitter una de las mayores redes sociales, cuyo lema es ”Es lo que está pasando” escoge a Cassandra como base de datos principal, en el 2016 ya contaba con más de 140 millones de usuarios y alrededor de 400 millones de tweets diarios. Para la gestión de todos estos datos Twitter diseña su arquitectura según el siguiente esquema[18].

Los datos llegan a través del stream api, y pasan una fase de transformación, donde al mismo tiempo en ser escritos en la base de datos también son leídos, esto es posible gracias a que los datos son accedidos por una clave única, generada en la fase de ETL (extract, transform, load). Twitter utiliza

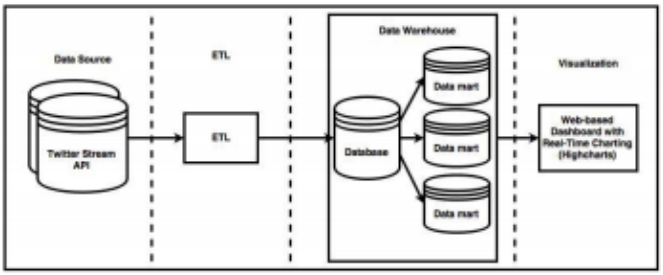

Fig. 7. Arquitectura Data WareHose Twitter [18]

Cassandra para la gestión de usuarios y su interacción, así como para la lectura de la mayoria de datos como los ”trending topics”, también para la gestión de tweets que anteriormente se utilizaba MySQL, pero debido a la internacionalización de la red social y que el tráfico en ella crecía exponencialmente se decidió el uso de Cassandra[19].

B. Netflix

Netflix, es la mayor empresa que ofrece contenido audiovisual bajo demanda por streaming. Cuenta con alrededor de 2500 almacenes de datos repartidos en todo el mundo, almacenando alrededor de 420TB de información y sobre un trillón de peticiones por día, ya sea de escritura y de lectura. Esta empresa ha basado su éxito gracias a la gestión del Big data, ya que series como House of Cards fueron producidas como el resultado de estudios de datos que fueron almacenando durante años de sus clientes. A día de hoy Netflix almacena toda la información posible, des del historial de visualización de un usuario hasta el minuto que un usuario pausa una película. Netflix comenzó utilizando Cassandra por las siguientes razones:

1) Cassandra tiene un buen soporte para modelado de datos de series de tiempo donde cada fila puede tener un número dinámico de columnas.
2) La relación de lectura y escritura de datos del historial de visualización es de aproximadamente 9: 1. Dado que Cassandra es muy eficiente con las escrituras, esta gran carga de trabajo de escritura es una buena opción para Cassandra.
3) Teniendo en cuenta el teorema de CAP, el equipo favorece la coherencia eventual sobre la pérdida de disponibilidad. Cassandra apoya esta compensación a través de una consistencia ajustable.

Esto permitió un escalado efectivo, cada usuario tenía una fila asignada y el número de columnas era dinámico. Cassandra pero no es un sistema perfecto y a medida que las columnas fueron creciendo en número y los usuarios transmitían más y más títulos, las lecturas se fueron haciendo cada vez más lentas. En las escrituras en cambio no se vieron afectadas. Ya que solo se escribía información nueva. Por eso fue necesaria la implementación de una caché, con la que Cassandra se adaptó de forma perfecta, ese es otro punto por la que escogieron Cassandra su integración con otras tecnologías. Además permitió la escalabilidad horizontal, agregando más

servidores, sin la necesidad de volver a fragmentar o reiniciar, para el caso. Por último mencionar la gran facilidad que ofrece Cassandra para el acceso a los datos, pudiendo acceder a datos únicos según su clave, que aprovechan los equipos de servicio para gestionar clientes de forma directa, o acceder a múltiples datos con filtros específicos y por regiones que aprovechan el equipo de marketing y producción para ofrecer producir contenido personificado para regiones o personas de una edad en concreto, todos eso gracias a su lenguaje de consultas CQL[20].

Aparte de estas dos grandes compañías, que han implementado de forma interna Apache Cassandra. Hay otras muchas grandes empresas que lo han hecho a través de empresas de soporte profesional como la de DataStax, que es la que oficialmente da soporte a Apacha Cassandra, teniendo como socios a empresas como HBO, Sony(Play Station), VISA. Todas ellas gestionan grandes volúmenes de datos, necesitando baja latencia de acceso y escritura que gracias a Cassandra lo pueden conseguir[21].

VII. Comparativa

No cabe duda que Cassandra es una de las bases de datos más usada actualmente pero no es la única. Existen varias bases de datos que si bien difieren en ciertas características con Cassandra son igual de usadas en distintos ámbitos o proyectos como pueden ser MongoDB o Hbase. Algunas de las características de Cassandra[22] son su almacenamiento en forma de columnas que ofrece velocidad a la hora de almacenar y buscar estos datos, su arquitectura en forma de "anillo" en el que cada nodo es tratado igual y gracia a esto se puede llegar a un quórum para ciertas prácticas, una distribución alta que permite un gran despliegue en países y zonas geográficamente separadas, herramientas avanzadas de lectura, escritura y consistencia de datos y gran tolerancia a los fallos. En cambio MongoDB es una base de datos con almacenamiento en formato de documentos. Estos datos semiestructurados normalmente en formato JSON, le otorgan una flexibilidad grande y le permite tener una naturaleza jerárquica, es decir, es posible generar un archivo JSON dentro de otro JSON y crear una jerarquía que facilite su almacenamiento y otorgue velocidad a la hora de realizar búsquedas. Usa un lenguaje propio, MongoDB query language (MQL), adaptado a la hora de buscar más fácilmente archivos JSON. Al igual que Cassandra, MongoDB proviene de una gran disponibilidad, lograda a través de réplicas que cuentan con características como redundancia de datos y conmutación por error automática, y gran escalabilidad horizontal.

Apache Hbase es un modelo de base de datos incluido en el Proyecto Apache Hadoop que usa el HDFS (Hadoop distributed file system) lo que le hace popular para trabajar con proyectos que usan Big Data. Ofrece una escalabilidad igual a las dos bases de datos anteriores y una tolerancia a fallos superior a estas y sus datos son almacenados en tablas que consisten en columnas y filas donde cada una de estas filas tiene un identificador único.

Para deducir que base de datos se ha de usar en un proyecto, se ha de comparar cada una de estas características y prestaciones. Un punto importante es el DbaaS, en el que todo el trabajo de administración de la base de datos se encarga a un administrador de la empresa distribuidora dando así más tiempo al proyecto a dedicarse al desarrollo de su producto. Cassandra y MongoDB tienen servicios DbaaS pero Hbase no posee ninguno. La forma de almacenar los datos también es importante. Mientras Cassandra y HBase usan un almacenamiento por columnas que mejora la velocidad de búsqueda y almacenamiento, MongoDB usa un sistema de almacenamiento por documentos. El lenguaje también juega un papel importante en la selección y en esto, las 3 bases de datos difieren entre sí. Mientras Cassandra usa su lenguaje CQL basado en SQL, MongoDB usa un lenguaje propio que también puede ser adaptado desde SQL[23] mientras que por el contrario Hbase no da soporte a lenguajes de búsqueda si no usa Java que es en lo que está basado Apache Hadoop.

Nombre de la BD	Cassandra	MongoDB	HBase
Arquitectura	Por columnas	Por documentos	Por columnas
Propietario y desarrollador	Apache Software Foundation	MongoDB, Inc.	Apache Software Foundation
Replicación	Masterless Ring	Replicación Master-Slave	Replicación Master-Slave
Lenguaje de programación (Código base)	Java	C++	Java
Casos de uso populares	Sistemas de mensajería, webs de comercio electrónico, aplicaciones siempre activas, detección de fraude para bancos	Gestión de datos de productos, sistemas de gestión de contenidos, IoT, análisis en tiempo real	Análisis de registros en línea, Hadoop, MapReduce, Aplicaciones con grandes cargas de escrituras
DbaaS	InstaClustr Cassandras as a Service, DataStax Database as a Service	MongoDB Atlas, mLab MongoDB, ScaleGrid MongoDB Hosting	Ninguno

Fig. 8. Tabla con las características principales

Para continuar esta comparación se usarán los pruebas y datos obtenidos en el siguiente estudio[24]. Se compara Cassandra, MongoDB y Hbase usando la herramienta YCSB (Yahoo Cloud Serving Benchmark). Con esta herramienta, se pasan 6 diferentes cargas de trabajo con las que se obtendrán los datos siguientes. Estas cargas son:

- Carga A: Carga compuesta de 50 por ciento lecturas y 50 por ciento escrituras
- Carga B: Carga compuesta de 95 por ciento lecturas y 5 por ciento escrituras
- Carga C: Carga totalmente compuesta por lecturas
- Carga D: Carga donde se buscan los últimos elementos guardados
- Carga E: Carga que busca un pequeño rango de valores en vez de un dato concreto
- Carga F: Carga en la que lee un dato, lo modifica y lo vuelve a escribir en la BD

En este experimento se han usado un millón de datos cada uno con un tamaño de 1kb. Cada uno de estos compuestos por 10 campos y cada uno de estos campos de un tamaño de 100 bytes.

Se obtiene los siguientes resultados:

- En la carga A, carga pesada en actualizaciones de datos, se observa como Hbase es la que mejor rinde gracias su diseño optimizado para la escritura de grandes cantidades

de datos.

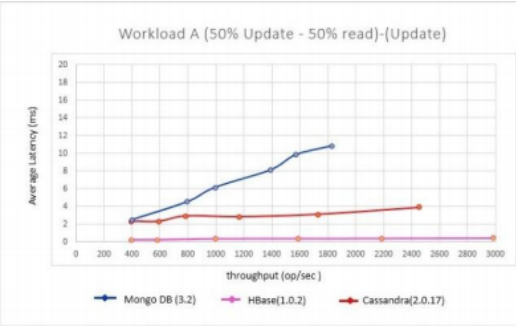

Fig. 9. Tabla con el resultado del workload A[24]

- En la carga B, MongoDB es superior gracias a su soporte de memoria mapeada en caché lo que contribuye a un aumento en el throughput y baja latencia comparada con las otras dos bases de datos.

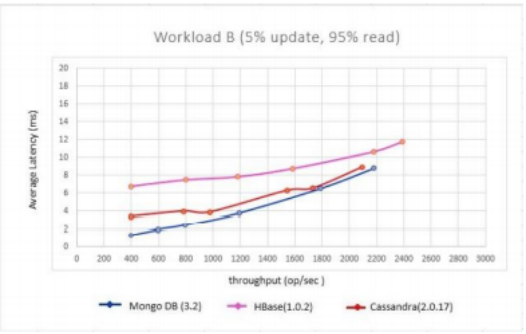

Fig. 10. Tabla con el resultado del workload B[24]

- En la carga C, se puede ver como Cassandra y MongoDB rinden de una manera casi igual. MongoDB toma la delantera al inicio de la carga pero se iguala a Cassandra cuando el throughput aumenta.

Fig. 11. Tabla con el resultado del workload C[24]

- En las cargas D y E, se cargan un millón de datos en cada base. El resultado obtenido es que Cassandra es la más rápida en tema de throughput con casi una carga de 19000 por segundo. Esto es debido a que se actualiza los datos en memoria mientras simultáneamente son escritos en el disco.

Fig. 12. Tabla con el resultado del workload D y E[24]

- En la carga F, se ve como tanto Hbase como Cassandra han rendido mejor en comparación con MongoDB confirmando de nuevo como estas dos bases de datos rinden mejor con un throughput alto.

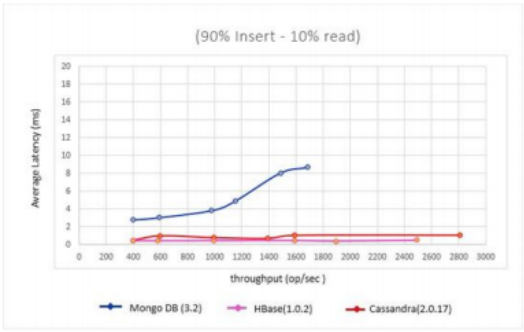

Fig. 13. Tabla con el resultado del workload F[24]

VIII. Conclusión y planes de futuro

Como se puede ver después de todo este estudio y, en especial, la comparativa anteriormente descrita, Cassandra rinde mejor en entornos donde se tenga que manipular grandes cantidades de datos. Es decir, en situaciones donde el throughput sea mayor de lo normal o donde se sepa que la escalabilidad del proyecto puede llegar a suponer un problema. Hoy en día, con la necesidad de manejar Big Data, Cassandra se ha hecho un hueco no tan solo en el mercado sino en varios proyectos activos. Si bien hay otras bases de datos como MongoDB, donde prima la lectura, o HBase, donde su diseño ayuda en operaciones de escritura masiva donde la cantidad de datos no suponga un problema, Cassandra es una buena opción por su almacenamiento basado en columnas que favorece una mejora en la velocidad de búsqueda de datos.

Gracias a su diseño de arquitectura se prioriza la alta disponibilidad sin un único punto de fallo antes que la consistencia, que es configurable a diferentes niveles, en escrituras y lecturas, como hemos visto en el estudio. Dado a su gran cantidad de datos, las estrategias de replicación y su organización de tokens, y rangos como el particionamiento son claves para que el sistema pueda permitirse escalar de

forma elástica, aumentando de forma lineal su rendimiento en relación con la cantidad de nodos que compongan el sistema.

REFERENCIAS Y BIBLIOGRAFIA

[1] Avinash Lakshman and Prashant Malik: 'Cassandra - A Decentralized Structured Storage System', *Facebook*, 2006

[2] Knowledge Base of Relational and NoSQL Database Management Systems, https://db-engines.com/en/ranking, *DB-Engines Ranking*, 2020

[3] NoSQL Ranking, https://stackoverkill.com/ranking/sql-nosql.html, *Stack-Overkill*, 2020

[4] Introducing Apache Cassandra 4.0 Beta: Battle Tested From Day One , https://cassandra.apache.org/blog/2020/07/20/apache-cassandra-4-0-beta1.html, *The Apache Cassandra Community* , 2020

[5] AWS : '¿Qué es NoSQL?',https://aws.amazon.com/es/nosql/, *Amazon*, 2020

[6] Hewitt, E. (2010). Cassandra: the definitive guide. " O'Reilly Media, Inc.".

[7] Maitrey J. Soparia: 'Apache Cassandra (Distributed Hash Table)', *Indiana University Bloomington*

[8] José Miguel Rojas Gonzales: 'Análisis comparativo de bases de datos relacionales y no relacionales', Universidad politécnica de Madrid.

[9] Juan José López Roldán: 'Cassandra NoSQL', 2018

[10] Emili Calonge Sotomayor, 'Base de dades basades en columnes', UPC, 2010

[11] Panagiotis Garefalakis,Panagiotis Papadopoulos, Ioannis Manousakis, Kostas Magoutis:'Strengthening Consistency in the Cassandra Distributed Key-Value Store'

[12] Piyusg Rana 'Cassandra Data Modeling: Primary, Clustering, Partition, and Compound Keys', 2016

[13] Patrick McFadin 'The most important thing to know in Cassandra data modeling: The primary key', DataStax, 2016

[14] Understanding Hinted Handoff (in Cassandra 0.8), https://www.datastax.com/blog/understanding-hinted-handoff-cassandra-08, *Jonathan Ellis*, 2011

[15] Cattell, R.: 'Scalable SQL and NoSQL data stores', 2011

[16] Osman, R., Piazzolla, P. :'Modelling replication in NoSQL Datastores in International Conference on Quantitative Evaluation of Systems', 2014

[17] Apache Cassandra : 'Documentación oficial sección Hardware',https://cassandra.apache.org/doc/latest/operating/hardware.html, 2020

[18] Muh. Rafif Murazza and Arif Nurwidyantoro : 'Cassandra and SQL database comparison for near real-time Twitter data warehouse',

[19] Oussalah, M., Bhat, F., Challis, K., Schnier, T. (2013). A software architecture for Twitter collection, search and geolocation services. Knowledge-Based Systems, 37, 105-120.

[20] Ketan Duvedi, Jihnua Li, Dhruv Garg, Philip Fisher-Ogden: 'Scaling Time Series Data Storage',https://netflixtechblog.com/scaling-time-series-data-storage-part-i-ec2b6d44ba39, 2018

[21] DataStax : 'Enterprise Succes' (2020), https://www.datastax.com/enterprise-success

[22] Evan Klein: 'Cassandra vs. MongoDB vs. Hbase: A Comparison of NoSQL Databases', https://logz.io/blog/nosql-database-comparison/, 2020

[23] MongoDB, Inc, https://docs.mongodb.com/manual/reference/sql-comparison/, 2020

[24] Ali Hammood, Murat Saran: 'A Comparison Of NoSQL Database Systems: A Study On MongoDB, Apache Hbase, And Apache Cassandra', https://www.researchgate.net, 2016

[25] Manuel Rubio: 'El Teorema CAP', https://altenwald.org/2017/05/23/teorema-cap/, AltenWald, 2017