Comparación de rendimiento de bases de datos NoSQL en modo pseudodistribuido: Casandra, MongoDB y Redis

Artículo · Septiembre 2015

4 autores, entre ellos:

Tiroshan Madushanka
Instituto Nacional de Informática
7 PUBLICACIONES 14 CITAS
VER EL PERFIL

Laksheen Mendis
Universidad de Moratuwa
2 PUBLICACIONES 11 CITACIONES
VER EL PERFIL

Comparación de rendimiento de bases de datos NoSQL en Modo pseudodistribuido: Cassandra, MongoDB y Redis

Kumarasinghe CU, Liyanage KLDU ,
Madushanka WAT y Mendis RACL Departamento
de Ingeniería y Ciencias de la Computación, Universidad de Moratuwa, Sri Lanka Correo
electrónico:{chamathk.10, tiroshan.10, dananji.10 & cresclux.10}@cse.mrt.ac.lk

Resumen: comprender el rendimiento de las bases de datos NoSQL es esencial para seleccionar la base de datos correcta para una aplicación específica. Este artículo describe en detalle y muestra los resultados de experimentos realizados sobre cómo se desempeña cada base de datos mencionada en el título con referencia a diferentes cargas de trabajo y diferentes simulaciones de concurrencia en modo pseudodistribuido.

I. INTRODUCCIÓN

Un problema

Las bases de datos NoSQL prometen un rendimiento más rápido y eficiente que el RDBMS heredado. Esto es más significativo en casos de uso que involucran Big Data. Seleccionar la base de datos correcta para la aplicación es un problema importante para los desarrolladores de aplicaciones, ya que el rendimiento de las bases de datos NoSQL varía según su categoría e implementación.

B. Antecedentes

Muchos arquitectos de sistemas y administradores de TI requieren comparaciones entre bases de datos NoSQL en sus entornos de desarrollo o producción utilizando datos e interacciones frecuentes de los usuarios, que representan la carga de trabajo que utilizarán en la nueva aplicación o sistema. Otra forma de entender las compensaciones del desempeño es definir puntos de referencia individuales bajo diferentes cargas de trabajo. Algunas de las bases de datos NoSQL tienen herramientas integradas para el análisis del rendimiento. Este tipo de análisis de desempeño no sería útil en un escenario de toma de decisiones como el descrito anteriormente. Es importante comprender las fortalezas y debilidades de cada base de datos en comparación con otras.

Por lo tanto, en este estudio se utiliza YCSB (Yahoo Cloud Serving Benchmark) como herramienta.

II. YCSB (PREFERENCIA DE SERVICIO EN LA NUBE DE YAHOO)

Tradicionalmente, ha habido una serie de puntos de referencia para bases de datos relacionales, pero estos no se pueden utilizar para comparar
las bases de datos NoSQL, que son no relacionales y vienen en diferentes categorías (por ejemplo: almacén de documentos, almacén de columnas, valor clave, gráfico, etc.).

Por lo tanto, el marco de evaluación comparativa de código abierto YCSB diseñado por Yahoo Labs intenta aliviar ese problema proporcionando un marco genérico para inducir la carga en un almacén de datos.
Por otro lado, este marco viene con interfaces para poblar y enfatizar muchos sistemas populares de servicio de datos, como
como; Cassandra, GemFire, HBase, HyperTable, DynamoDB, MongoDB, Redis, Voldemort, OrientDB, etc.

Hay muchas otras herramientas de evaluación comparativa alternativas para cada base de datos específica, por ejemplo, MongoDB tiene una herramienta incorporada llamada mongoperf que puede brindar resultados de la velocidad de las operaciones de lectura/escritura y otra herramienta llamada mongo-perf que puede usarse para comparar el rendimiento del rendimiento. por la cantidad de subprocesos y cómo ocurren las búsquedas del disco mientras se ejecutan las pruebas.

A. Configuración Un

resumen de los pasos seguidos durante la configuración de YCSB, para comparar las bases de datos NoSQL. • El proyecto GitHub está disponible

para YCSB y para compilarlo desde el código fuente, es necesario clonar el proyecto. • YCSB usa Apache Maven como herramienta de compilación
y ejecutar una limpieza en el paquete compilaría YCSB. • Hay un Shell de YCSB donde está Es posible ejecutar comandos u otra opción
sería escribir un script de shell para automatizar las pruebas. • También es importante especificar el número de registros que se insertarán para la prueba, el número de operaciones en las
pruebas y el número de subprocesos a los que se ejecutará la operación. ser ejecutado.

Esto se puede hacer tanto para cargar como para ejecutar las pruebas.

B. Cargas de trabajo

Las cargas de trabajo constan de seis tipos diferentes de cargas de trabajo para simular diferentes contextos de aplicación.

- Carga de trabajo A:

  Carga de trabajo para actualizaciones intensas. En esta carga de trabajo el 50% de las operaciones son lecturas y el 50% son escrituras. Esta carga de trabajo simula la grabación y actualización de sesiones de usuarios en aplicaciones web.

- Carga de trabajo B:

  Carga de trabajo que tiene más operaciones de lectura que escrituras. En esta carga de trabajo el 95% de las operaciones son lecturas y el 5% restante son escrituras. Esta carga de trabajo simula la generación de informes en una solución ERP que tiene grandes volúmenes de datos. En este escenario, la mayoría de las operaciones serían para

leer los datos de las tablas y escribir un documento resumido en otra tabla.

• Carga de trabajo C:

La carga de trabajo solo tiene operaciones de lectura. Esta carga de trabajo simula la visualización de libros y revistas en línea.

• Carga de trabajo D:

La carga de trabajo inserta nuevos registros y los registros insertados más recientemente son los más populares. Esta carga de trabajo simula las actualizaciones de estado de los usuarios, donde las personas quieren leer las últimas novedades.

• Carga de trabajo E:

La carga de trabajo consulta rangos cortos de registros, en lugar de registros individuales. Esta carga de trabajo simula conversaciones encadenadas, donde cada escaneo es para las publicaciones en un hilo determinado (suponiendo que los hilos estén agrupados por ID de hilo).

• Carga de trabajo F:

Carga de trabajo donde el cliente leerá un registro, lo modificará y escribirá los cambios. Esta carga de trabajo simula una base de datos de usuarios, donde el usuario lee y modifica los registros de usuario o una situación que registra la actividad del usuario.

III. INTERNOS DE LA BASE DE DATOS

A. Casandra

1) Introducción: Cassandra es un sistema de gestión de bases de datos de código abierto, que está diseñado para manejar grandes cantidades de datos con licencia Apache. Esto se puede utilizar en muchos servidores básicos que proporcionan alta disponibilidad y ningún punto único de falla. Este es un proyecto nacido en Facebook y construido sobre Amazons Dynamo y Googles Big Table. Cassandra pretende funcionar sobre una infraestructura de cientos de nodos. A esta escala, los componentes grandes y pequeños tienden a fallar, pero Cassandra logra mantener un estado persistente que asegura la confiabilidad y escalabilidad de los sistemas de software que dependen de este servicio. Cassandra aborda el problema de las fallas empleando un sistema distribuido de igual a igual en nodos homogéneos donde los datos se distribuyen entre todos los nodos del clúster.

2) Arquitectura: en Cassandra, los datos se indexan y escriben en una estructura en memoria llamada memtable. Un registro de confirmación escrito secuencialmente en cada nodo captura la actividad de escritura para garantizar la durabilidad de los datos. Cuando la estructura de la memoria está llena, los datos se escriben en el disco en un archivo de datos SSTable. Casandra es una

Familia de columnas. Estas bases de datos están diseñadas para almacenar datos sin esquemas. En un almacén clave-valor, todos los datos constan de una clave indexada y un valor. Está construido en base al modelo Key-Value. Por lo tanto, al explorar Cassandra, nos encontramos con los términos Familia de columnas y Espacio clave. Un espacio de claves es análogo a una base de datos, mientras que una familia de columnas corresponde a una tabla. Por lo tanto, un espacio de claves contiene una o más familias de columnas. En Cassandra cada familia de columnas es

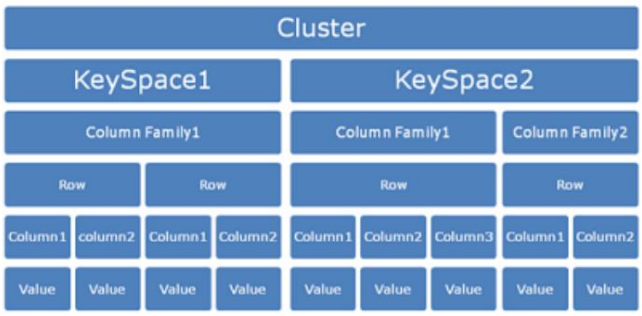

Fig. 1. Modelo de datos de Cassandra

almacenado en un archivo separado que se ordena en una fila. Una familia de columnas es un contenedor para una colección de filas.

3) Cassandra Internals: Cassandra utiliza un árbol de fusión estructurado en registros como estructura de almacenamiento. Cassandra evita leer antes de escribir. Leer antes de escribir conduce a cachés corruptos y aumenta los requisitos de E/S. Por lo tanto, para evitar esto, el motor de almacenamiento agrupa las inserciones/ actualizaciones que se realizarán y luego agrega solo las partes actualizadas de una fila. Cassandra nunca relee/reescribe y nunca sobrescribe los datos existentes.

Cassandra crea subdirectorios para las tablas dentro de cada directorio de espacio de claves. Esto permite vincular simbólicamente una tabla a un controlador físico. Esto proporciona la capacidad de mover una mesa a medios más rápidos como SSD para obtener un mejor rendimiento.

Con respecto a las propiedades ACID en las bases de datos SQL, las bases de datos NoSQL también mantienen algunas propiedades similares a ellas.
Como base de datos NoSQL, Cassandra está diseñada para optimizar la disponibilidad y la tolerancia de partición. Por lo tanto, la coherencia con respecto a Cassandra se refiere a cómo se actualiza y sincroniza una fila de datos en todas sus réplicas. Cassandra procesa datos en varias etapas, en la ruta de escritura, comenzando con el registro inmediato de una escritura y terminando con la compactación.

La compactación es la operación de fusionar múltiples SSTables en una sola nueva.

Indexación en Cassandra: internamente, un índice de Cassandra es una partición de datos. Cassandra usa un índice para extraer los registros en cuestión. Esta partición es la unidad de replicación en Cassandra.
Las particiones se distribuyen mediante hash de la clave principal o de la clave candidata. Cada nodo indexa sus propios datos. Esto evita visitar varios nodos al recuperar datos.

Al igual que otras bases de datos relacionales, Cassandra también tiene que actualizar sus índices. Cuando se actualiza una columna, el índice también se actualiza. Cassandra elimina los índices obsoletos correspondientes. Si una lectura ve una entrada de índice obsoleta antes de que la compactación la purgue, el hilo del lector la invalida.

B.MongoDB

1) Introducción: MongoDB es una base de datos NoSQL orientada a documentos que utiliza BSON, un lenguaje similar a JSON para recuperar y actualizar la base de datos. Es una de las principales bases de datos NoSQL que maneja grandes colecciones de datos (big data).
MongoDB está escrito en C++ y es un proyecto de código abierto que utiliza pthreads para funciones relacionadas con la concurrencia. MongoDB

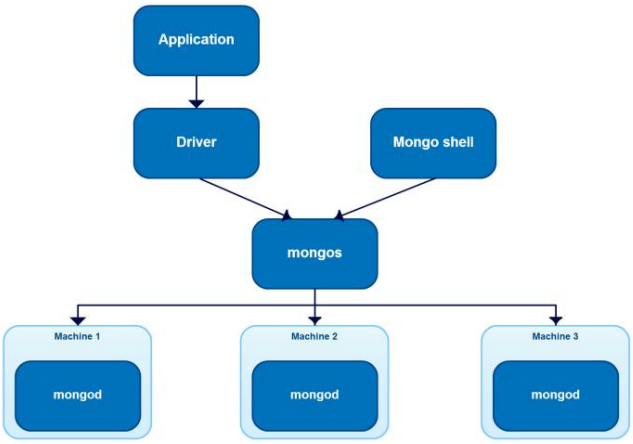

Fig. 2. Arquitectura de MongoDB

La atención se centra en la coherencia de los datos y la tolerancia de las particiones a expensas de la disponibilidad según el teorema CAP de Brewers.

Hay muchas empresas destacadas que utilizan MongoDB como base de datos back-end. Se trata de SAP, eBay, SourceForge, Fouresquare e incluso el supercolisionador del CERN.

2) Arquitectura: Hay tres partes principales en la base de datos Mon-goDB: • Servidor MongoDB

comúnmente conocido como demonio mongod.

Es el proceso principal que maneja las solicitudes de datos, administra el formato de los datos y realiza operaciones de administración en segundo plano. Puede haber muchos demonios mongod ejecutándose como instancias secundarias primarias.

- MongoDB mongos es el servicio de enrutamiento. Este proceso enruta información y datos en el cluster.
- MongoDB shell es la interfaz interactiva. Al utilizar JavaScript para ordenar, el desarrollador puede examinar los resultados de las consultas y comprobar los casos de prueba.

3) Componentes internos de MongoDB: en MongoDB, todos los documentos se insertan con un hash único. Esto se utiliza al recuperar registros durante la consulta. Hay varios tipos de otros índices, a saber: • Índices únicos: se aplican a un campo de un documento

para rechazar otros documentos duplicados. • Índices compuestos: se aplican a varios campos de un documento

para rechazar otros documentos duplicados. • Índices de matriz: todas las matrices son indexado por esto en un documento para

mejorar el rendimiento • Índices TTL: donde es necesario eliminar documentos después de un cierto tiempo para salir

- Otros índices: índices geoespaciales, índices dispersos y Índices de búsqueda de texto

Para la optimización de consultas, las cachés de MongoDB ejecutan previamente planes de consulta, además de ejecutar otros planes alternativos y seleccionan el plan con el mejor tiempo de respuesta. Los resultados de los planes alternativos también se almacenan en la memoria caché para referencia futura.

MongoDB realiza operaciones con datos existentes y almacena sus datos en bloques contiguos. Al actualizar los datos en el lugar, se pueden reducir las operaciones de E/S al no requerir el movimiento de datos a otras ubicaciones. Es un hecho conocido que la lectura desde la memoria es varios miles de veces más rápida que la lectura desde el disco; MongoDB utiliza esto a su favor al mantener en la memoria los índices a los que se accede con frecuencia.

MongoDB lee y escribe en la RAM donde el sistema operativo realiza el mapeo de memoria a la memoria virtual.

C. Regreso

1) Introducción: Redis (Servicio de diccionario remoto) es un servidor de base de datos NoSQL de código abierto con licencia BSD. Redis es un almacén avanzado de clave-valor y se considera un servidor de estructura de datos, ya que las claves pueden contener muchos de los tipos de datos fundamentales, como cadenas, listas, conjuntos, conjuntos ordenados y hashes. Redis trabaja con un conjunto de datos en memoria para reducir la sobrecarga de acceso a la memoria no volátil y, por lo tanto, se supone que mantiene todo el espacio clave en la memoria durante las operaciones. Cuando finalice, los datos se borrarán de la memoria. Pero dependiendo del caso de uso, los datos se pueden conservar tomando instantáneas de los datos y volcandolos en el disco periódicamente o manteniendo un registro de todas las operaciones de solo agregar. Redis se implementa utilizando ANSI C y funciona en sistemas POSIX (Linux, BSD, OS X y Solaris) sin dependencias externas.

2) Arquitectura: la arquitectura de Redis se basa en el enfoque BASE (básicamente disponible, de estado suave y eventualmente consistente) y abandona el enfoque ACID (atomicidad, consistencia, durabilidad y aislamiento) en RDBMS. Redis se basa en la arquitectura cliente/servidor y consta de los siguientes componentes: • Servidor Redis • Servidores Redis Réplica (opcional) • Cliente

Redis

3) Componentes internos de Redis DB: Redis admite funciones avanzadas como conjuntos ordenados y muchas otras operaciones atómicas complejas, ya que está en la memoria y es de un solo subproceso. El concepto de memoria se implementa utilizando el concepto de memoria virtual, que es una característica muy importante de la mayoría de los sistemas operativos modernos. Pero por razones de eficiencia, Redis no utiliza las funciones de memoria virtual proporcionadas por el sistema operativo y, en cambio, implementa su propio sistema llamado Redis Virtual Memory.

El objetivo de Redis Virtual Memory (VM) es intercambiar datos a los que se accede con poca frecuencia de la RAM al disco, sin cambiar drásticamente las características de rendimiento de la base de datos y al mismo tiempo admitir conjuntos de datos que son más grandes que la memoria principal.

El fundamento de la memoria virtual de Redis es el siguiente: • Una sola página, administrada por el sistema operativo, ocupa 4 kB. • El valor de una única clave de Redis puede afectar a muchas páginas diferentes, incluso si la clave es lo suficientemente pequeña como para caber en una sola página.

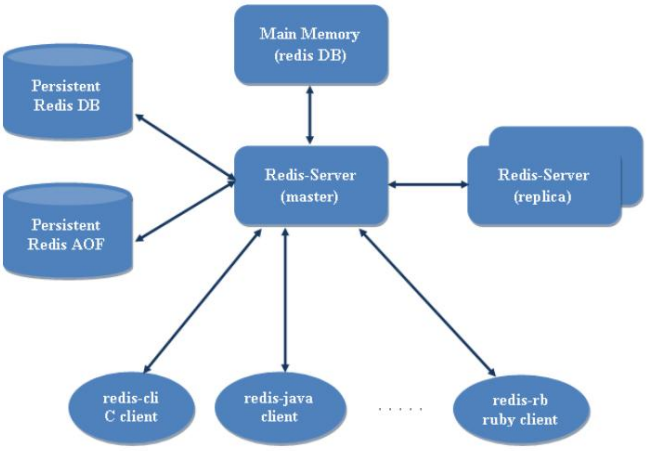

Fig. 3. Arquitectura de la base de datos Redis

• Los objetos de Redis pueden tener un orden de magnitud mayor en la RAM que cuando se almacenan en el disco. Por lo tanto, si se utilizan las funciones de memoria virtual del sistema operativo, el sistema operativo necesitaría realizar un orden de magnitud más de E/S en comparación con una implementación personalizada de memoria virtual de Redis.

En Redis, la memoria virtual no tiene memoria, transfiere valores que pertenecen a claves que no se usaron recientemente de la memoria al disco. Cuando un comando de Redis intenta acceder a una clave que se intercambia, se vuelve a cargar en la memoria.

En la implementación de Redis VM, todos los valores se intercambiarán, pero las claves y la tabla hash de nivel superior seguirán usando RAM, así como el "mapa de bits de la tabla de páginas", que es una matriz de bits en la memoria de Redis que contiene información sobre páginas usadas/libres en el archivo de intercambio.

4) Implementación de memoria virtual: Redis VM almacena la última vez que se accedió a cada objeto y mantiene un archivo de intercambio que se divide en páginas de tamaño configurable, con la tabla de asignación de páginas almacenada en la memoria.

Cuando Redis VM se queda sin memoria y selecciona el objeto con el factor de intercambiabilidad más alto como el objeto que se intercambiará en el disco.

Además, Redis mantiene un grupo de subprocesos de E/S que son los únicos responsables de cargar valores desde el disco a la RAM. Cuando llega una solicitud, se lee el comando y se examina la lista de claves. Si alguna de las claves se ha intercambiado al disco, el cliente se suspende temporalmente mientras un trabajo de E/S está en cola. Una vez cargadas todas las claves que necesita un cliente determinado, el cliente reanuda la ejecución del comando.

IV. CONFIGURACIÓN Y ESPECIFICACIÓN DEL SISTEMA

El objetivo final de este proceso de evaluación comparativa es comparar el rendimiento de diferentes bases de datos NoSQL en varias instancias en modo pseudodistribuido. Si los experimentos se realizaron para diferentes bases de datos (es decir, Cassandra, MongoDB y Redis), en diferentes máquinas, los resultados dependerán de las especificaciones del sistema. Por tanto, no se puede comparar.

Por tanto, los experimentos se realizaron en una sola máquina.

A continuación se muestran las especificaciones:

• Procesador: Intel Core i7-3630QM • Velocidad de reloj: 2,40 GHz • Número de núcleos: 4
• Número de hilos: 8
• Conjunto de instrucciones: 64 bits • Memoria: 8061 MB • Sistema operativo: Ubuntu 14.04.1 LTS • Kernel: Linux 3.13.0-35-generic
• Java: OpenJDK 7 • Versión YCSB: 0.1.4 • Versión Cassandra: 2.0.10 • Versión de MongoDB: 2.7.4 • Versión de Redis: 2.8.15

V. RESULTADOS EXPERIMENTALES

Esta sección incluye gráficos que muestran el rendimiento de las 6 cargas de trabajo proporcionadas por YCSB. Para cada carga de trabajo se consideraron 3 instancias con 1000, 5000 y 10000 registros. Además, para cada tamaño de registro se han considerado 1,2,4 y 8 hilos.

A. Carga de trabajo A

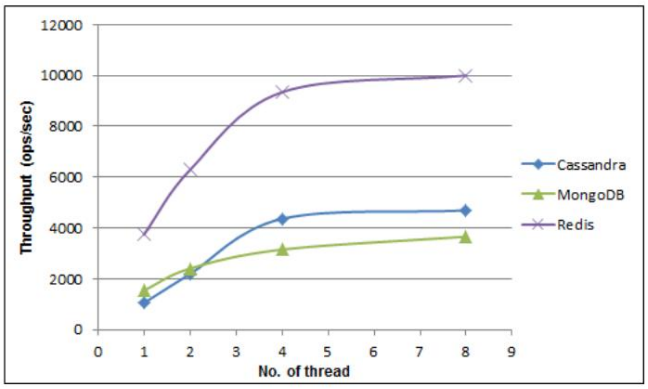

Fig. 4. Rendimiento con 1000 registros y 1000 operaciones

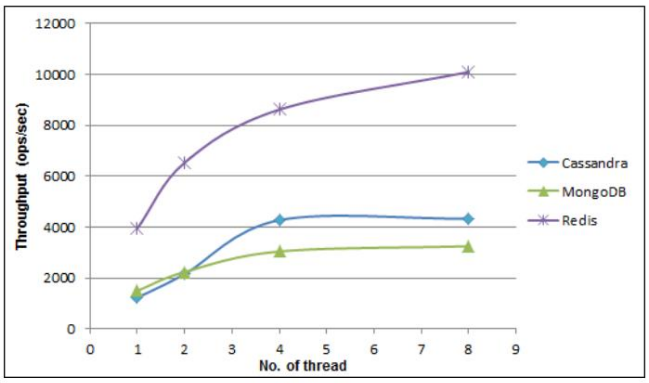

Fig. 5. Rendimiento con 5000 registros y 1000 operaciones

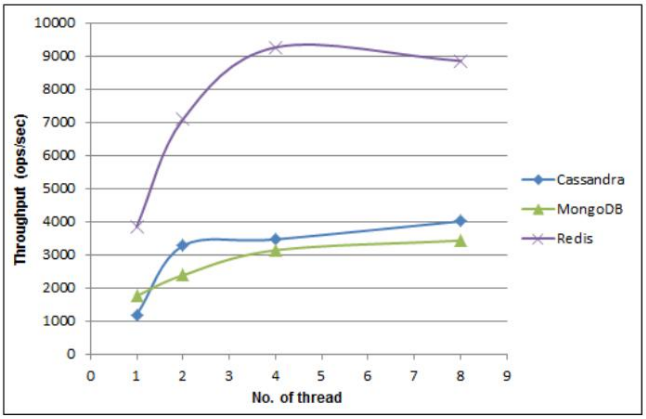

Fig. 6. Rendimiento con 10000 registros y 1000 operaciones

Para la gran carga de trabajo de actualización, se puede ver que Redis tiene el doble de rendimiento que las bases de datos MongoDB y Cassandra. Con baja concurrencia (1, 2 subprocesos), MongoDB y Cassandra tienen un rendimiento similar, pero Cassandra tiene un rendimiento ligeramente mejor cuando aumenta la cantidad de subprocesos y la cantidad de registros.

Esto puede deberse al hecho de que los servicios de replicación de datos proporcionados por las bases de datos MongoDB y Cassandra no existen en modo pseudodistribuido, por lo que utilizan el mapeo de memoria en varios nodos para aumentar el rendimiento.

También es notable que la ganancia de velocidad después de 4 subprocesos no aumenta rápidamente a pesar de que el hardware permite ejecutar 8 subprocesos simultáneamente.

B. Carga de trabajo B

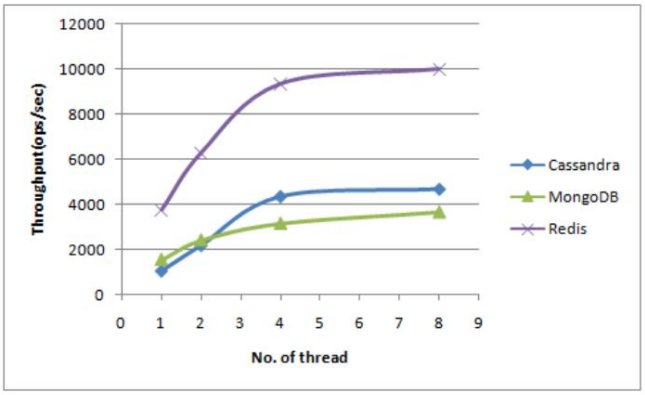

Fig. 7. Rendimiento con 1000 registros y 1000 operaciones

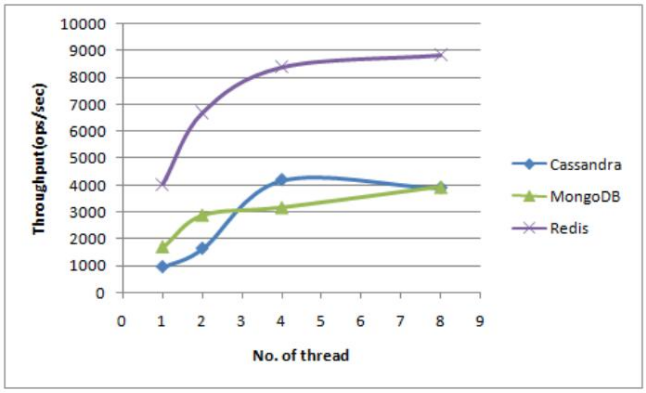

Fig. 8. Rendimiento con 5000 registros y 1000 operaciones

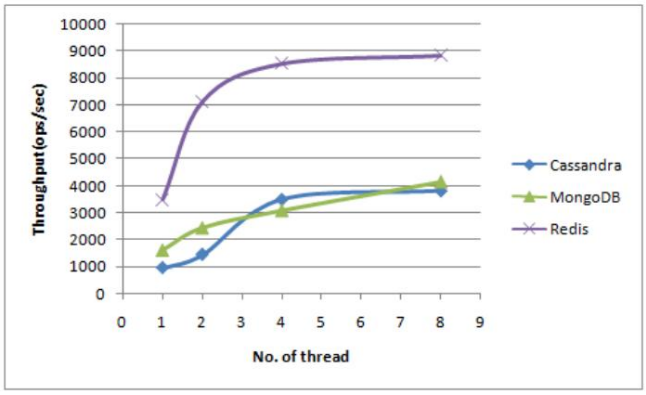

Fig. 9. Rendimiento con 10.000 registros y 1.000 operaciones

Los resultados de esta carga de trabajo de mayor cantidad de lecturas son similares a la carga de trabajo de actualización pesada, pero MongoDB y Cassandra tienen un rendimiento idéntico en 8 subprocesos (número máximo de subprocesos ejecutables simultáneamente por el hardware). MongoDB y Cassandra intercambian lugares en 2 y 4 subprocesos.

C. Carga de trabajo C

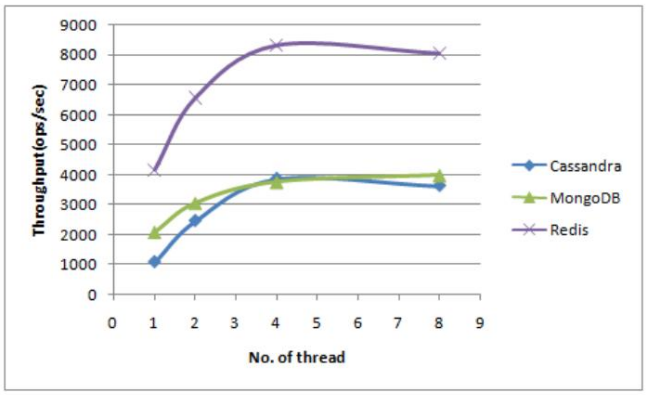

Fig. 10. Rendimiento con 1000 registros y 1000 operaciones

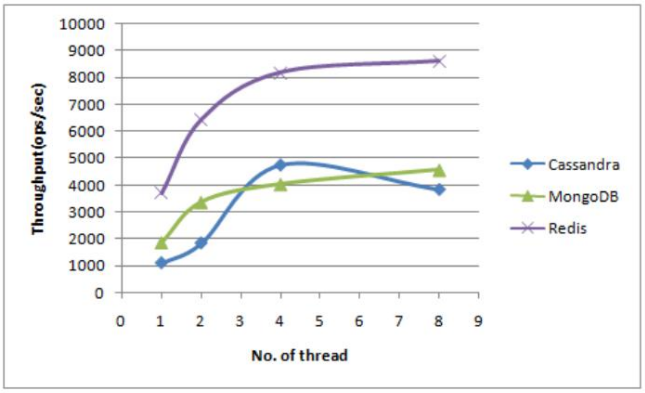

Fig. 11. Rendimiento con 5000 registros y 1000 operaciones

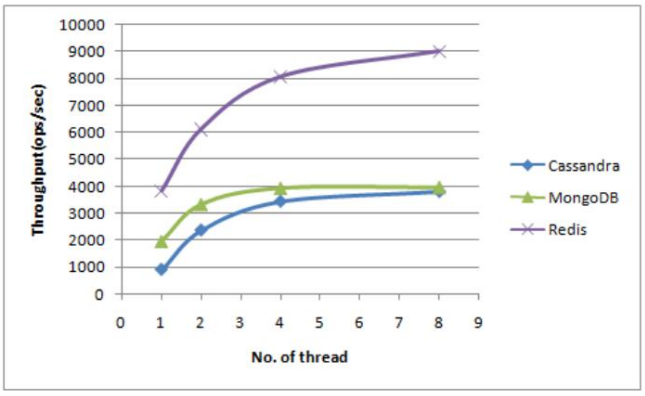

Fig. 12. Rendimiento con 10.000 registros y 1.000 operaciones

Cassandra sufre una latencia de lectura alta, lo que reduce el rendimiento general. Esto también se puede ver en MongoDB.

En los gráficos del Workload C, que tiene un 100% de operaciones de lectura, esto se puede ver claramente. Mientras que el rendimiento general de Redis es mayor que el de MongoDB y Cassandra. Con el incremento del número de subprocesos y el número de registros, el rendimiento general de Cassandra y MongoDB converge.

D. Carga de trabajo D

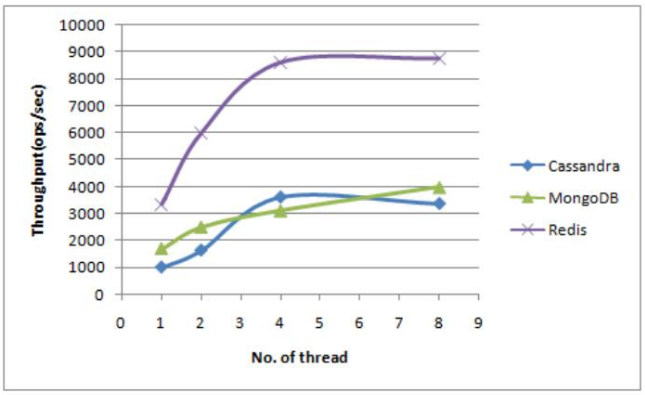

Fig. 13. Rendimiento con 1000 registros y 1000 operaciones

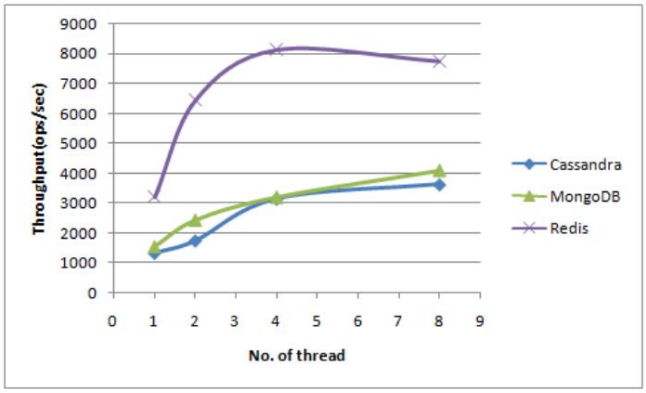

Fig. 14. Rendimiento con 5000 registros y 1000 operaciones

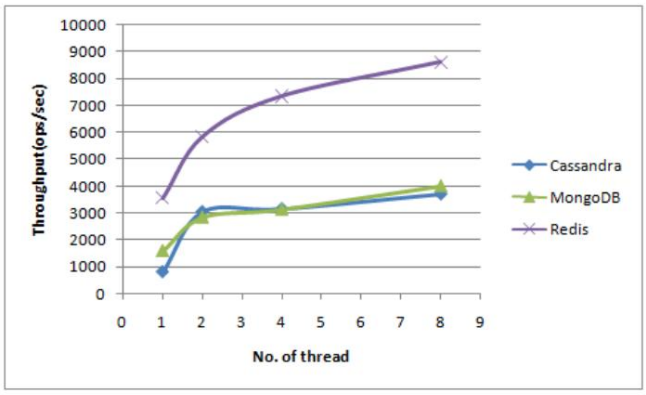

Fig. 15. Rendimiento con 10.000 registros y 1.000 operaciones

Se puede ver que ninguna de las bases de datos utiliza mecanismos de almacenamiento en caché para optimizar los últimos datos escritos para recuperarlos rápidamente, ya que el rendimiento apenas cambia con respecto a la carga de trabajo A en cada una de las bases de datos.

E. Carga de trabajo E

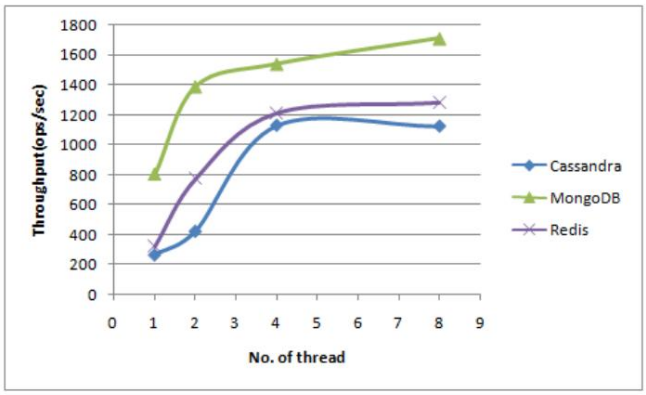

Fig. 16. Rendimiento con 1000 registros y 1000 operaciones

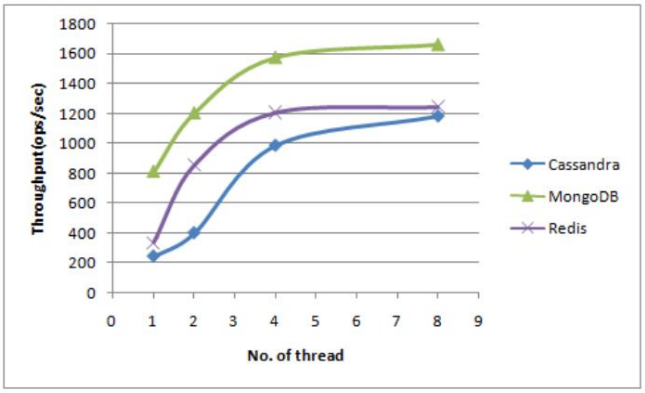

Fig. 17. Rendimiento con 5000 registros y 1000 operaciones

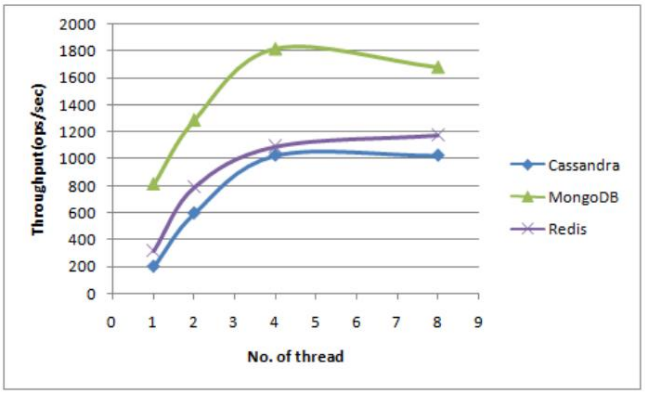

Fig. 18. Rendimiento con 10.000 registros y 1.000 operaciones

Para consultas de corto alcance, se puede ver que MongoDB supera a Cassandra y, sorprendentemente, también a Redis.
Cassandra y Redis tienen rendimientos similares, pero Redis tiene un rendimiento ligeramente mejor, aunque parece degradarse con la cantidad de registros en la base de datos.

F. Carga de trabajo F

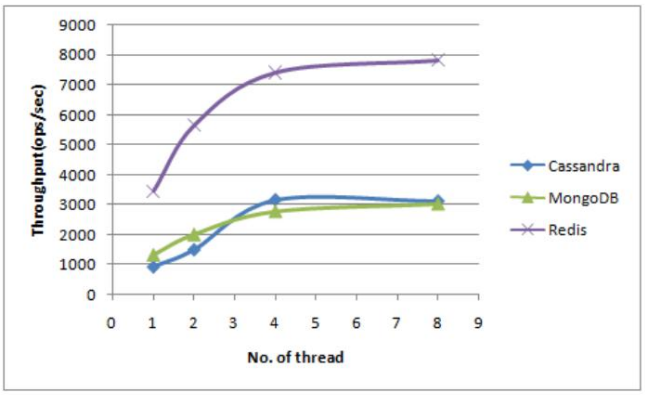

Fig. 19. Rendimiento con 1000 registros y 1000 operaciones

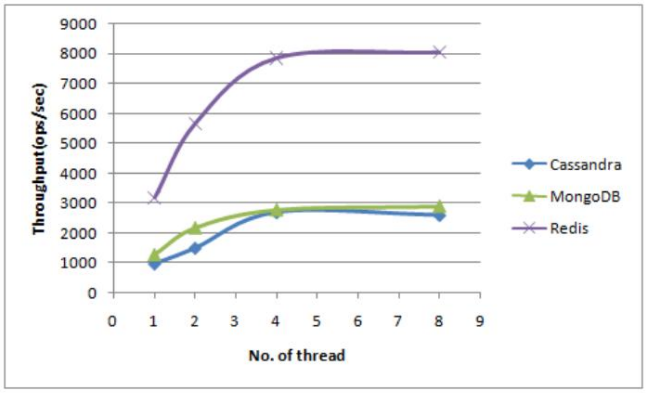

Fig. 20. Rendimiento con 5000 registros y 1000 operaciones

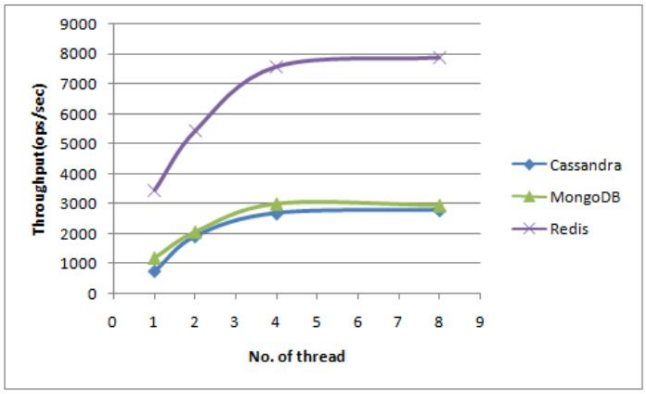

Fig. 21. Rendimiento con 10.000 registros y 1.000 operaciones

En la carga de trabajo de lectura, modificación y escritura, Redis tiene casi 4 veces el rendimiento de MongoDB y Cassandra. Esto podría deberse al hecho de que Redis realiza sus operaciones de memoria en la memoria antes de escribir en el disco, mientras que MongoDB escribe datos en el disco.

G. Rendimiento y latencia con respecto al número de subprocesos

Se llevó a cabo un experimento para identificar el comportamiento de las 3 bases de datos para 1000000 registros con 1-50 subprocesos en 50 instancias. A continuación se muestran gráficos resumidos de latencia de lectura, latencia de actualización y rendimiento.

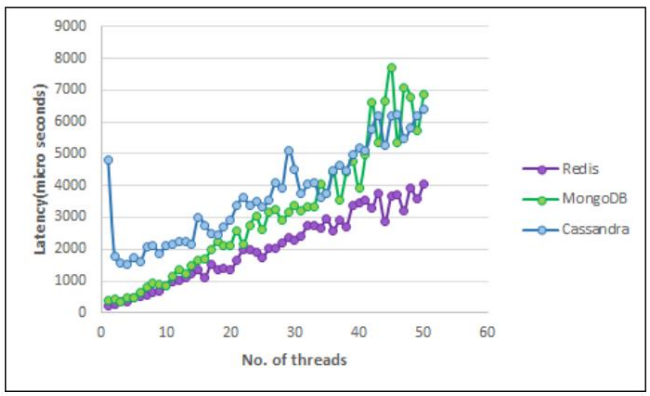

Fig. 22. Lectura de latencia para 100.000 registros

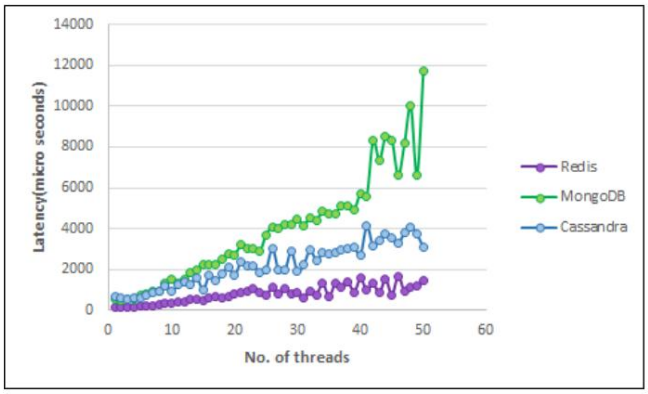

Fig. 23. Latencia de actualización para 100000 registros

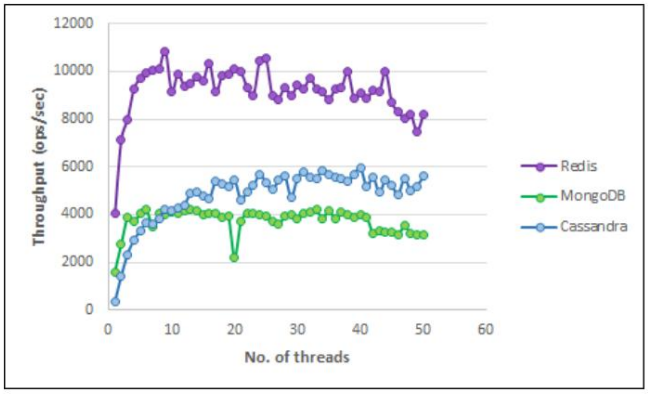

Fig. 24. Rendimiento para 100.000 registros

Se puede ver que sólo en Cassandra el rendimiento aumenta con el número de hilos. En Redis, el rendimiento disminuye con el número de subprocesos y en MongoDB el El rendimiento permanece casi sin cambios hasta aproximadamente 40 subprocesos. después de lo cual dimite.
Dado que Cassandra optimiza las operaciones de escritura en lugar de las de lectura, completa casi el 100% de las operaciones de actualización. inmediatamente, por lo tanto, YCSB configura Cassandra para proporcionar consistencia débil por defecto. Casandra es más adecuada para sistemas distribuidos, mientras que el experimento se llevó a cabo en un modo pseudodistribuido. Por lo tanto, la latencia de lectura de Cassandra es superior a MongoDB. En modo distribuido Cassandra puede leer localmente sin incurrir en latencia entre centros de datos, lo que mostrará una latencia de lectura más baja que MongoDB.

VI. CONCLUSION

En resumen se puede concluir que en pseudodistribuidos El modo Redis tiene un mejor rendimiento para aplicaciones que no requiere consultas de rango y los desarrolladores pueden elegir Redis sobre Cassandra y MongoDB para aplicaciones que tienen tales requisitos. Se pueden hacer más análisis sobre el rendimiento. de estas bases de datos para un gran número (millones) de registros y operaciones para comparar aún más las bases de datos. YCSB puede ser utilizado como herramienta para este propósito.

VII. REFERENCIAS

[1] Cooper, BF, Silberstein, A., Tam, E., Ramakrishnan, R., y Sears, R. Evaluación comparativa de sistemas de servicios en la nube con ycsb. En Actas del primer simposio de ACM sobre computación en la nube (Nueva York, NY, EE. UU., 2010), SoCC '10, ACM, págs.143-154.

[2] Cargas de trabajo principales. Recuperado de septiembre a julio [EN LÍNEA] 2014 Disponible: https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads

[3] Documentación de NoSQL Apache Cassandra — Planeta Casandra. 2014. [EN LÍNEA] Disponible: http://planetCassandra.org/documentation/.

[4] mongod. Consultado en septiembre de 2014 [EN LÍNEA] Disponible: http://docs.mongodb.org/manual/reference/program/mongod/bi n. mongod

[5] Índices y MongoDB. Proyecto de documentación de MongoDB, [ONLINE] Disponible: https: //docs.mongodb.org/master/MongoDB-indexes-guide.pdf

[6] SCons: una herramienta de construcción de software. Recuperado en septiembre-Febrero, 2014 [EN LÍNEA] Disponible: http://www.scons.org/

[7] Paksula, M., Introducción al almacenamiento de datos en Redis, una base de datos de valores clave persistente y rápida, en Actas de AMICT 2010-2011 Avances en métodos de tecnología de la información y la comunicación (Helsinki, Finlandia, 2011),AMICT,pp.39-47.

[8] Carga de trabajo en ejecución. Recuperado en agosto de 2014 [EN LÍNEA] Disponible: https://github.com/brianfrankcooper/YCSB/wiki/Running-a-Workload

[9] mongodb/mongo GitHub. diosb/mongo GitHub. 2014. mi- [EN LÍNEA] Disponible: https://github.com/mongodb/mongo.