Escuela Superior Politécnica del Litoral

Facultad de Ingeniería en Electricidad y Computación

Evaluación de rendimiento de sistemas de almacenamiento clave-valor utilizando reproducción de cargas de trabajo reales

Trabajo de titulación previo a la obtención del Título de

"Magíster en Ciencias de la Computación"

Presentado por:

Edwin Federico Boza Gaibor

Guayaquil - Ecuador

2018

Resumen

Los sistemas de almacenamiento de tipo clave-valor son un componente importante de la arquitectura de aplicaciones en la nube. Estos sistemas, que son un subconjunto de las bases de dato NoSQL, generalmente no soportan transacciones ACID, por lo tanto, su utilización diverge de las aplicaciones comunes para bases de datos tradicionales y no están bien representadas por herramientas estándar de evaluación para bases de datos, tales como TPC-C. En este contexto, YCSB ha surgido como el estándar de facto para evaluación de sistemas de almacenamiento en la nube, brindando soporte para una amplia variedad de cargas de trabajo sintéticas.

Sin embargo, una correcta evaluación de rendimiento requiere ser complementada con la reproducción de cargas de trabajo realistas, obtenidas a partir de sistemas en producción. No obstante, no existe una herramienta públicamente disponible que permita reproducir trazas reales contra sistemas de almacenamiento en la nube, lo cual es perjudicial para la comunidad científica. Como resultado, otros investigadores han desarrollado herramientas ad-hoc de reproducción de cargas de trabajo, que utilizan modelos de operación no documentados, haciendo imposible replicar, o incluso analizar, sus resultados.

En el presente trabajo, se propone un modelo de reproducción de cargas de

trabajo reales, aplicable para sistemas de almacenamiento de tipo clave-valor y se describe KV-replay, que es una herramienta de código abierto, para reproducción de cargas de trabajo, que implementa el modelo propuesto.

A través de un análisis experimental con memorias caché, que son una implementación práctica de sistemas de clave valor, se muestra que utilizar cargas de trabajo sintéticas conlleva a errores de evaluación significativos, de hasta un un 33 % en la tasa de fallos para tamaños de caché pequeños y una sobre-estimación de hasta el 18 % en la aceleración del sistema. Las evaluaciones realizadas muestran además que la herramienta KV-replay es precisa en ordenamiento y temporalidad de los eventos generados, además de ser lo suficientemente rápida para no afectar el rendimiento de las pruebas de evaluación.

Índice general

Capítulo 1

Introducción y objetivos

1.1. Antecedentes y justificación

Los sistemas de almacenamiento clave-valor son un componente importante de las aplicaciones en la nube. Estas bases de datos, que son un subconjunto de aquellas conocidas como NoSQL, no soportan transacciones ACID, por lo tanto, sus aplicaciones difieren de aquellas que utilizan bases de datos tradicionales y sus cargas de trabajo no son representadas adecuadamente por *benchmarks* como TPC-C y TPC-H (Poess y Floyd, 2000).

En este contexto, YCSB (Cooper et al., 2010) ha surgido como el estándar *de-facto* para la evaluación de sistemas de almacenamiento en la nube, proveyendo un conjunto de *macro-benchmarks* sintéticos que dan soporte a una amplia variedad de cargas de trabajo específicas para sistemas de bases de datos NoSQL.

A pesar de que los *macro-benchmarks* sintéticos son importantes en las evaluaciones de sistemas de almacenamiento, algunos componentes de estos son sensibles a dimensiones no reproducidas por los generadores sintéticos. Por ejemplo, la *lo-*

calidad temporal es una dimensión importante no reproducida por modelos como el Modelo de Referencias Independientes (IRM) (Vanichpun y Makowski, 2004b), que es utilizado por muchos de los generadores sintéticos, incluyendo YCSB. Otras dimensiones de la carga de trabajo, no reproducidas por YCSB incluyen los inter-arribos y fluctuaciones en intensidad de los pedidos (Pitchumani et al., 2015) y el tamaño de los valores. Por este motivo, se considera buena práctica complementar las evaluaciones de rendimiento basadas en cargas de trabajo sintéticas con evaluaciones que utilicen la reproducción de trazas de cargas de trabajo (Traeger et al., 2008).

Sin embargo, no existe una herramienta disponible públicamente que pueda reproducir cargas de trabajo reales para sistemas clave-valor. Esto es dañino para la comunidad de investigadores, debido a la importancia de la reproducción de cargas de trabajo reales para la evaluación completa y adecuada de sistemas de almacenamiento.

Como resultado, otros investigadores han desarrollado herramientas *ad-hoc* de reproducción de trazas, de fuente cerrada, que utilizan modelos de reproducción no documentados, imposibilitando la replicación de sus resultados. Lo que es más importante, desarrollar una herramienta propia es una medida propensa a errores, ya que elegir e implementar el modelo de reproducción adecuado no es una tarea trivial (Pereira et al., 2016; Schroeder et al., 2006; Traeger et al., 2008).

Existen preguntas que deben ser respondidas, pero que reciben poca atención de parte de los investigadores al momento de implementar herramientas *ad-hoc* de reproducción de trazas. Por ejemplo: ¿se debería usar un reproductor basado en compilación o basado en eventos (Pereira et al., 2016)?, ¿se debe elegir un modelo de reproducción abierto o cerrado (Schroeder et al., 2006)?, ¿se debe priorizar la

exactitud de los tiempos de arribo o garantizar el orden de los eventos (Weiss et al., 2013; Zhu et al., 2005)?, ¿cómo se puede reproducir cargas de trabajo pesadas que no pueden ser manejadas por una instancia única del reproductor? ¿cómo se puede escalar (temporal y espacialmente) la carga de trabajo?.

Diferentes enfoques pueden conducir a diferentes resultados de rendimiento (Pereira et al., 2016; Schroeder et al., 2006; Traeger et al., 2008). Además, las decisiones de diseño e implementación de reproductores *ad-hoc* raramente están bien documentadas, haciendo difícil entender los resultados obtenidos. Por ejemplo, muchos reproductores de traza implementan solo la política de ejecución a máxima velocidad; sin embargo, aunque esta política es útil para pruebas de estrés de sistemas, si se utiliza con un modelo de reproducción abierto no considera ninguna posible retroalimentación entre las solicitudes (Pereira et al., 2016). Cuando un trabajo de investigación utiliza una herramienta para reproducir una traza a máxima velocidad, sin indicar el modelo de reproducción, resulta imposible valorar el impacto de la metodología de reproducción en los resultados obtenidos.

Adicionalmente, a diferencia de sistemas de bases de datos tradicionales para las que existe un API común (ODBC), los sistemas actuales de clave-valor tienen sus propias APIs propietarias, haciendo difícil construir una herramienta de evaluación que pueda ejecutarse en una amplia variedad de estos productos.

1.2. Objetivo general

Desarrollar un marco de trabajo que permita una correcta evaluación de rendimiento de sistemas de almacenamiento de tipo clave-valor, mediante el uso de la técnica de reproducción de cargas de trabajo reales.

1.3. Objetivos específicos

Los objetivos específicos del presente trabajo son:

- Proponer un modelo de reproducción de trazas apropiado para sistemas de almacenamiento de tipo clave-valor.
- Implementar el modelo propuesto en una herramienta de código abierto.
- Evaluar la aplicación del modelo, considerando la precisión, rapidez y utilidad del modelo y herramienta.

Capítulo 2

Marco teórico

2.1. Bases de datos NoSQL

Los sistemas de bases de datos NoSQL[1], también conocidas como *No Solo SQL*, comprenden un conjunto de sistemas de almacenamiento que tienen como objetivo resolver los problemas de escalabilidad y rendimiento de las aplicaciones web modernas y del procesamiente de grandes datos, por lo cual, sus principales características radican en que son sistemas distribuidos, escalables horizontalmente y no relacionales.

Se puede clasificar a las bases de datos NoSQL, dependiendo de la estructura de los datos que pueden almacenar, lo que a su vez determina el campo de aplicación de la misma. Se pueden diferenciar cuatro tipos de sistemas de almacenamiento NoSQL: clave-valor, basada en columnas, basadas en documentos, de grafos y

[1]El término NoSQL fue utilizado por Carlo Strozzi en 1998 para nombrar un sistema de bases de datos relacionales basado en archivos que no utiliza el lenguaje de consultas SQL (Strozzi, 1998). La misma expresión ha sido adoptada a partir del 2009, reinterpretándola como "*Not Only SQL*", para referirse a un creciente conjunto de sistemas de almacenamiento que se alejan de los sistemas relacionales tradicionales con el fin de ofrecer mayor escalabilidad y rendimiento (Evans, 2009).

multimodales (que combinan uno o varios de los otros modelos de datos)(Haseeb y Pattun, 2017).

2.2. Sistemas de almacenamiento de clave-valor

Los sistemas de almacenamiento de clave-valor son un tipo de bases de datos NoSQL que implementa el modelo de arreglos asociativos, también conocido como diccionario. En estos sistemas, los registros son almacenados y recuperados utilizando una clave única; la clave tiene uno o más valores asociados a ella (ver Figura 2.1).

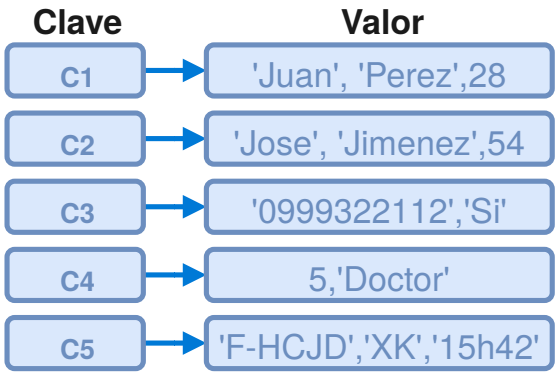

Figura 2.1: Vista lógica de los datos almacenados en un sistema de almacenamiento de tipo clave-valor.

Existen muchas implementaciones de sistemas de almacenamiento clave-valor, que difieren en sus garantías de consistencia, persistencia y rendimiento.

Algunos usos comunes de almacenamientos de clave-valor incluyen: almacenamiento en caché de objetos para mejorar latencia de acceso, contadores de seguimiento e implementaciones de canales publicador-subscriptor (Atikoglu et al., 2012; Gravell, 2011).

Debido a su alta escalabilidad y rendimiento, los sistemas de almacenamiento

de clave-valor se han convertido en componentes críticos de grandes sistemas en la nube y de Big Data. Por este motivo, son un tópico popular de estudios y continuamente se desarrollan esfuerzos significativos para mejorar su rendimiento, escalabilidad, costo y consumo de poder (Atikoglu et al., 2012). Para que tales esfuerzos sean efectivos, se requieren métodos de evaluación adecuados, incluyendo micro y macro-benchmarks, y herramientas para reproducción de cargas de trabajo reales.

2.3. Cachés distribuidas

Los sistemas de memoria caché están incluidos en muchas aplicaciones modernas para proveer un acceso rápido a objetos, reduciendo tanto la carga sobre la capa de almacenamiento persistente, como la latencia percibida por el cliente cuando accede a un objeto. Una caché distribuida está desplegada sobre múltiples servidores para permitir niveles elevados de escalabilidad, confiabilidad y disponibilidad, a la vez que se incrementa la capacidad transaccional.

Un algoritmo de caché es un método heurístico que trata de mantener en la memoria caché aquellos objetos que son propensos a ser accedidos. De esta manera se reducen la cantidad de requerimientos que son enviados a la capa de almacenamiento permanente. Un *hit* o acierto es el evento cuando el objeto requerido se encuentra y es servido directamente desde la memoria caches. El rendimiento de las memorias cachés, con frecuencia es medido con los tiempos de respuesta (latencia) y la tasa de aciertos (*hit rate*).

Los sistemas de cachés son un caso de aplicación natural de las bases de datos clave-valor, a las que se les adiciona los correspondientes algoritmos de admisión

y/o desalojo. Para aplicaciones en la nube, se utilizan capas de sistemas de almacenamiento caché implementadas con almacenamientos de clave-valor distribuidos tales como Redis y Memcached.

2.4. YCSB

El Yahoo Cloud Serving Benchmark (YCSB) (Cooper et al., 2010) es el estándar de facto para evaluación de rendimiento de sistemas de almacenamiento en la nube y es ampliamente utilizado en la industria y en la academia (Pitchumani et al., 2015).

Esta herramienta tiene una arquitectura modular, lo que facilita la inclusión de nuevas bases de datos y nuevas cargas de trabajo.

La Figura 2.2 muestra el diseño de YCSB, donde se aprecia que posee una capa extensible para conexiones a bases de datos; además de un paquete generador de cargas de trabajo con varias opciones de parametrización (`CoreWorkload`).

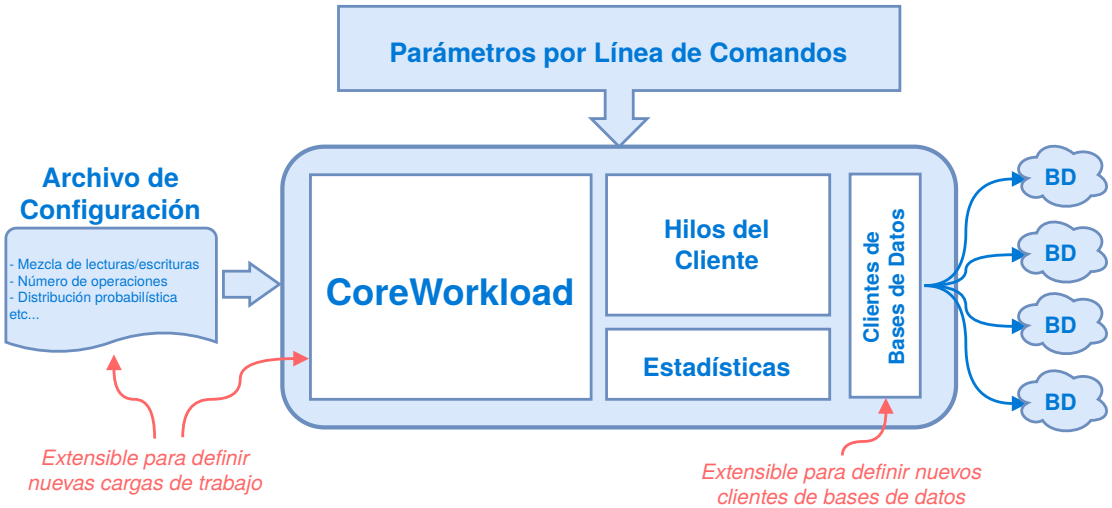

Figura 2.2: Arquitectura extensible del Yahoo Cloud Serving Benchmark (YCSB).

2.4.1. Extensión de conectividad a bases de datos en YCSB

La capa de conexión a bases de datos ofrece soporte para una lista creciente de sistemas de bases de dato NoSQL. Más aún, se facilita la creación de nuevos *plugins* para conectar nuevas bases de datos[2].

Para agregar conectividad a una nueva base de datos a YCSB, se debe extender la clase `com.yahoo.ycsb.DB` e implementar los siguientes métodos, que corresponden a los comandos de una base de datos: *read, scan, update, insert and delete.*

2.4.2. Extensión de cargas de trabajo en YCSB

YCSB posee seis cargas de trabajo sintéticas preconfiguradas, con variaciones en la proporción de lecturas/escrituras inspiradas por casos de uso comunes de servicios en la nube. Nuevas cargas de trabajo pueden ser creadas mediante la configuración de la distribución probabilística (Uniforme, Zipfian, Latest o Multinomial) que genera las claves de los accesos, la proporción de las operaciones de cada tipo y el tamaño de los registros.

YCSB fue diseñado para pruebas de estrés, por lo que las operaciones son generadas a la máxima velocidad posible. Sin embargo, es posible implementar generadores de cargas de trabajo con diferentes características para permitir la evaluación de dimensiones adicionales[3].

[2]https://github.com/brianfrankcooper/YCSB/wiki/Adding-a-Database
[3]https://github.com/brianfrankcooper/YCSB/wiki/Implementing-New-Workloads

2.4.3. Limitaciones de YCSB

Además de no proveer soporte para la reproducción de cargas de trabajo reales, YCSB no modela dimensiones importantes de las cargas de trabajo, tal como se describe a continuación:

- YCSB emite eventos a una tasa constante (máxima velocidad). Sin embargo, tiempos de *interarribos* realistas son necesarios cuando la presencia de picos en la intensidad de lo arribos (*burstiness*) afecta el rendimiento (Al-Shishtawy y Vlassov, 2013; Pitchumani et al., 2015).

- La *localidad temporal* es otro factor que puede causar un impacto en el rendimiento de un sistema. Por ejemplo, es bien conocido que las memorias cachés (que son una aplicación común de los sistemas de almacenamiento de clave-valor) son afectadas por la localidad temporal (Abad et al., 2013; Bianchi et al., 2013; Traverso et al., 2013; Vanichpun y Makowski, 2004a) y no deben ser evaluadas únicamente utilizando el Modelo de Referencias Independientes (*Independent Reference Model - IRM*), en el que cada acceso a un objeto es independiente de los accesos anteriores (Abad et al., 2013; Traverso et al., 2013). A pesar de esto, YCSB utiliza este modelo[4].

- El *tamaño de los valores* también puede afectar el rendimiento, por ejemplo, Memcached tiene particiones (*slabs*) para diferentes rangos de tamaños. Si todos los valores son generados con el mismo tamaño, tal como lo hace YCSB, se incrementará la contención en el slab correspondiente.

[4]YCSB tiene una carga de trabajo que no utiliza IRM, llamada *Zipfian Latest*. Sin embargo, ésta provee una manera poco realista de simular localidad temporal y raramente ha sido utilizada en los artículos consultados.

Estas falencias de YCSB se pueden mitigar agregando soporte para reproducción de trazas cuyo formato contengas las dimensiones mencionadas.

Capítulo 3

Modelo

3.1. Tipo de reproductor

Los reproductores de trazas pueden ser **basados en compilación o en eventos** (Pereira et al., 2016). Aquellos basados en compilación crean un programa que emula la actividad capturada en la traza; por ejemplo, SWIM para Hadoop (Chen et al., 2011). Por otro lado, los reproductores basados en eventos son programas genéricos que repiten los eventos presentes en la traza; este tipo de reproductores son usados frecuentemente en sistemas de almacenamiento (Joukov et al., 2005; Mesnier et al., 2007; Pereira et al., 2016; Tarasov et al., 2012; Traeger et al., 2008; Weiss et al., 2013; Zhu et al., 2005).

El enfoque basado en eventos es el más adecuado para los sistemas de almacenamiento de tipo clave-valor, ya que la meta es brindar soporte para trazas de diferentes aplicaciones, mientras que los reproductores basados en compilación son más adecuados para *benchmarks* de aplicaciones específicas.

3.2. Ciclo de reproducción

Los generadores de cargas de trabajo pueden ser de ciclo **abierto o cerrado** (Pereira et al., 2016; Schroeder et al., 2006). En el modelo de ciclo abierto, los eventos generados son independientes unos de otros y pueden ser enviados incluso si los eventos previos no han sido completados aún. Mientras que en un generador de ciclo cerrado, uno o más clientes o hilos despachan eventos uno tras otro, con algún *think time* en medio; un evento es enviado solo después de que el evento previo ha sido completado. Adicionalmente, existe un modelo híbrido, o *parcialmente abierto*, en el que los clientes arriban al sistema en momentos diferentes y generan requerimientos durante una *sesión*; los eventos en una sesión son despachados en un modelo de ciclo cerrado. Para el caso de los reproductores de trazas, los modelos cerrados y parcialmente abiertos son equivalentes, ya que la diferencia entre ambos está dada por el momento en que arriban los clientes y esto está definido en la traza y no se requiere una generación por medio de un proceso estocástico de arribos.

3.3. Elección de modelos basados en la arquitectura

Un supuesto común en reproducción de trazas es que el tiempo de espera entre dos requerimientos de un cliente, está dado por el tiempo de respuesta del sistema más el *think time* del cliente. Una alternativa es ignorar los interarribos y simular los *think times* (Schroeder et al., 2006).

Benchmarks de bases de datos como TPC-C y TPC-H, usan un esquema basa-

do en compilación, de ciclo cerrado (Schroeder et al., 2006). Para RDBMSs, tiene sentido utilizar un enfoque basado en compilación y despachar consultas SQL complejas en lugar de enviar eventos que accedan registros individuales. Las consultas SQL son enviadas en un ciclo o bucle cerrado para modelar las dependencias en una transacción o secuencia de consultas.

Los sistemas de almacenamiento de clave-valor proveen modelos de consistencia más débiles que aquel provisto por bases de datos ACID y son utilizados para aplicaciones fuera del tradicional Online Transaction Processing (OLTP) o sistemas de soporte de decisiones. Ejemplos de aplicaciones soportadas por sistemas de almacenamiento de tipo clave-valor incluyen (Atikoglu et al., 2012; Cooper et al., 2010; Gravell, 2011; Stackoverflow.com, 2017):

- **Almacenamiento de objetos en caché**; e.g., una caché para perfiles de usuarios, donde los perfiles son construidos en otro sistema como Hadoop.
- **Etiquetado de fotos**, donde agregar una etiqueta es una actualización, sin embargo, la mayoría de las operaciones son lecturas de etiquetas.
- **Base de datos de usuario**, donde los usuarios lean y modifiquen sus registros o el mismo sistema almacene registros de la actividad de los usuarios.
- **Contadores de seguimiento**; e.g., contadores de interacción con el contenido (*user engagement*), como seguimiento de clics a objetos.
- **Almacenamiento de sesiones**, almacenando acciones en sesiones de usuarios.
- **Actualización de estados de usuario** en redes sociales; los usuarios actualizan su propio estado y leen las actualización de estado más recientes de

otros usuarios.

- **Canales de Publicación/Suscripción**; e.g., seguimiento de precios de acciones, donde los subscriptores de un canal pueden hacer el seguimiento al precio de una acción de la bolsa de valores.

Basados en las dinámicas de estas cargas de trabajo, se puede argumentar que, a menos que el análisis de la carga de trabajo sugiera lo contrario, un ciclo abierto puede ser usado para evaluar el rendimiento de sistemas de almacenamiento en la nube de tipo clave-valor. Intuitivamente, el modelo abierto es adecuado para muchas aplicaciones en la nube; ya sea porque captura bien el comportamiento del sistema (cada evento es independiente de los otros) o porque aún cuando existen unas pocas dependencias (principalmente en accesos desde una sesión cliente), el número de evento por sesión es pequeño y el tiempo entre las operaciones dependientes es dominado por el *think time* del usuario y no por el tiempo de respuesta del sistema. Por ejemplo, un usuario que etiqueta sus fotos, lo hará visualizándolas y manualmente agregando las etiquetas necesarias.

Considerando la traza de YouTube usada en la Sección 5.2: solo el 0,13 % y el 0,004 % de los eventos de un cliente individual tienen un tiempo de interarribo menor que la latencia máxima y promedio del sistema clave-valor observadas los experimentos llevados a cabo en este experimento, respectivamente. Para esta carga de trabajo, usar un modelo de ciclo cerrado (en el que los eventos provenientes del mismo cliente son despachados solo después de que los eventos previos de dicho cliente han finalizado) es indistinguible de la reproducción de la traza utilizando un modelo abierto.

A continuación, se describen tres arquitecturas específicas, pero comunes en

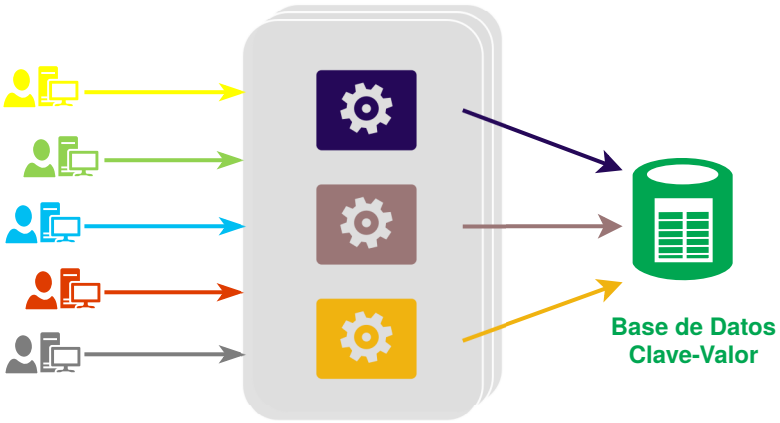

Figura 3.1: Primer ejemplo de arquitectura en la nube: Capa de microservicios.

aplicaciones en la nube actuales, y se discute qué modelo es apropiado para reproducir sus cargas de trabajo

Arquitectura en la nube 1: Frecuentemente, una capa de microservicios se sitúa entre el usuario final y la base de datos clave-valor. Los procesos de los microservicios se ejecutan en servidores de aplicación tales como *node.js* que utilizan llamadas *async* (sin bloqueos) para despachar los requerimientos hacia la capa de la base de datos. Una traza almacenada en esta configuración, podría sugerir un ciclo cerrado, donde existen pocos clientes enviando muchos requerimientos cada uno; sin embargo, estos requerimientos son independientes unos de otros, por lo que se sugiere utilizar un ciclo abierto para reproducir la traza.

Arquitectura en la nube 2: Para sistemas de almacenamiento que atienden directamente a usuarios finales, tales como Tarantool (`tarantool.org`), que combina un servidor de aplicaciones con una base de datos que puede ser consultada directamente por los clientes web, se puede seguir las indicaciones sugeridas por Schroeder et al. (Schroeder et al., 2006) para elegir el modelo de reproducción adecuado; esto es, identificar las sesiones por cliente y contar el número de requerimientos en cada sección. Un sistema parcialmente abierto se comporta similar

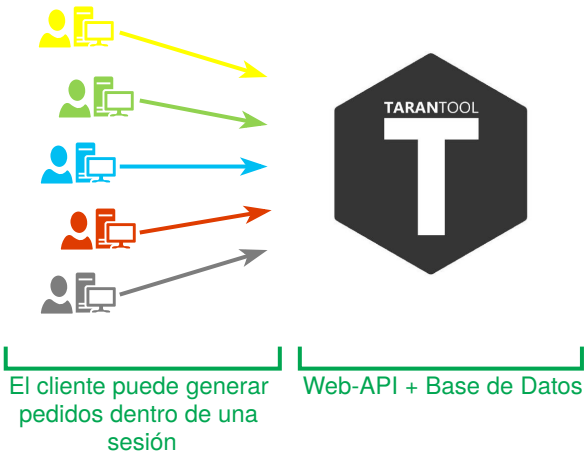

Figura 3.2: Segundo ejemplo de arquitectura en la nube: Base de datos de tipo clave-valor atendiendo directamente requerimientos de usuarios (e.g.: Tarantool)

a un sistema abierto cuando el número esperado de requerimientos por sesión es pequeño ($\leq$ 5 (Schroeder et al., 2006)) por lo que es seguro elegir un modelo de reproducción abierta. Para sesiones más largas, los requerimientos deben ser despachados utilizando un ciclo parcialmente abierto.

Arquitectura en la nube 3: En grandes granjas de sistemas de almacenamiento de tipo clave-valor, tiene sentido utilizar un *proxy* liviano y rápido, tal como Twemproxy[1], para consolidar y encaminar los requerimientos de los clientes y reducir la presión o carga en la conexión a la base de datos. El modelo correcto de reproducción de cargas de trabajo dependerá de cómo fue capturada la traza correspondiente. Si la traza fue grabada en la capa de almacenamiento clave-valor, entonces el comportamiento del sistema será mejor representado por un modelo de ciclo cerrado. Si, por el contrario, la traza fue capturada en la capa del proxy, entonces se deberían seguir las mismas indicaciones descritas en el ejemplo de la arquitectura de nube 2.

En resumen, la diferencia en las cargas de trabajo conduce a la utilización

[1] https://github.com/twitter/twemproxy

Figura 3.3: Tercer ejemplo de arquitectura en la nube: Consolidación de requerimientos mediante el uso de un servicio proxy ligero.

de diferentes modelos de reproducción. La implementaciones de reproductores de trazas tienden a utilizar el esquema de reproducción cerrado (Schroeder et al., 2006), debido a que es más fácil de implementar que un modelo abierto. Sin embargo, es recomendable utilizar un modelo abierto por defecto, pero dar a los usuarios la oportunidad de seleccionar el modelo cerrado cuando así lo crean conveniente.

Capítulo 4

Diseño e implementación

En este capítulo se describen las decisiones de diseño e implementación de una herramienta de reproducción de trazas de cargas de trabajo, de código abierto, llamada KV-replay.

4.1. Arquitectura

KV-replay fue implementado como una extensión de YCSB, principalmente con el fin de aprovechar sus *plugins* de conexión a bases de datos (para enero del 2017, YCSB contaba con clientes para 29 sistemas de almacenamiento en la nube, más un cliente de conexión ODBC).

Para implementar el modelo abierto de reproducción de trazas, se utilizó la clase de Java `ScheduledExecutorService`, la cual permite calendarizar la ejecución de eventos. El enfoque simplista de espera (sleeping) entre eventos no funciona cuando dos eventos (A y B) ocurren tan cercanos en el tiempo que B debe ser enviado antes de que la respuesta de A haya sido recibida (YCSB utiliza llamadas con bloqueo). Las evaluaciones muestran que el enfoque sugerido permite repro-

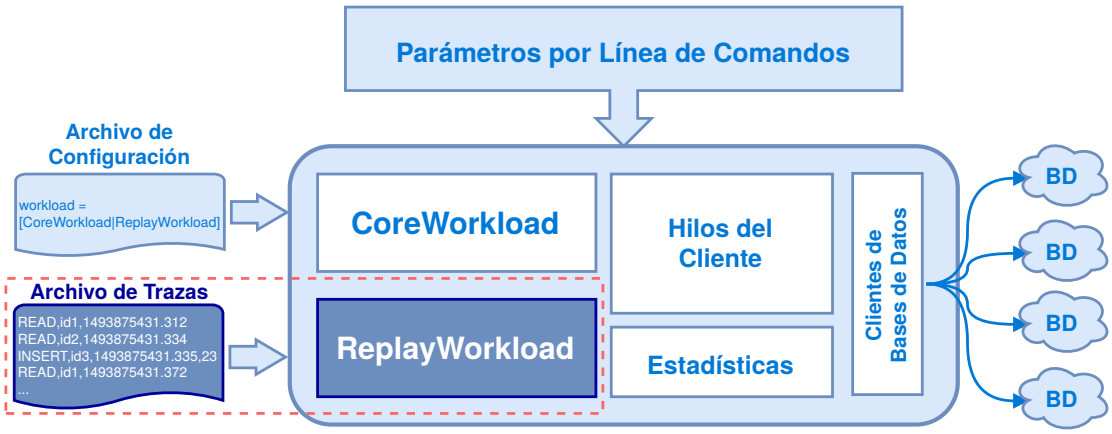

Figura 4.1: Arquitectura de KV-replay, como una extensión de YCSB. Los bloques oscuros indican el módulo que ha sido agregado.

ducir los tiempos de la traza con una precisión de milisegundos.

La Figura 4.1 muestra la arquitectura de KV-replay. A diferencia del módulo original `CoreWorkload`, que envía eventos generados de manera sintética, KV-replay utiliza el módulo `ReplaryWorkload` para reproducir los eventos almacenados en una traza provista por el usuario. El usuario puede seleccionar el tipo de carga de trabajo a utilizar (sintética o reproducción de traza) mediante un parámetro en el archivo de configuración.

Para alcanzar una alta tasa de envíos (throughput), el usuario puede usar varios clientes, cada uno con uno o más hilos de ejecución. Los clientes usan una barrera de inicio basada en tiempo para asegurar que se empieza el despacho de eventos al mismo tiempo (más detalles son provistos en la sección 5.4). Adicionalmente, si múltiples máquinas son utilizadas, sus relojes deben ser sincronizados antes de iniciar las evaluaciones.

Evaluar un sistema de almacenamiento de tipo clave-valor con KV-replay requiere ejecutar las siguientes fases:

1. La fase de carga de datos, donde se prepara la base de datos llenándola

con las claves presentes en la carga de trabajo a ser evaluada. Esta fase es **opcional**.

2. La fase de ejecución de la carga de trabajo, en la cual la traza es reproducida y se emiten los requerimientos correspondientes.

Se recomienda ejecutar KV-replay como un proceso en tiempo real para evitar problemas de multitarea apropiativa con otros procesos ejecutándose en la misma computadora.

4.2. Carga de trabajo

Formato de la traza. Los registros contienen los siguientes campos <Comando, ID del Ojeto, Estampa de Tiempo, Tamaño>. La estampa de tiempo es opcional, debido a que no es necesaria cuando se reproduce la traza a máxima velocidad. El tamaño del objeto también es opcional, dado que el usuario puede elegir usar el mismo tamaño (configurable) para todos los valores a lo largo de la ejecución.

Obtener una traza en este formato, es solo cuestión de reformatear una traza dada, mediante un script de pre-procesamiento.

El valor utilizado para las operaciones de insersión o actualización son generados aleatoriamente por KV-replay. Los valores no son incluidos en la traza debido a que esto podría hacer el archivo considerablemente grande, dificultando la compartición y distribución del mismo. Adicionalmente, el contenido de los valores no afecta los resultados de rendimiento de la mayoría de las pruebas.

Eventos perdidos o faltantes. Los métodos de captura de trazas pueden perder un pequeño porcentaje de los eventos (Traeger et al., 2008; Zhu et al., 2005). Esto

Tabla 4.1: Formato de los registro de la traza de entrada utilizada por KV-replay. Los campos opcionales están marcados con un asterisco (*).

Campo de la traza	**Descripción**
Comando	El comando que será ejecutado en la base de datos, tal como está definido por YCSB.
ID del objeto (clave)	El identificador del objeto (clave) apuntado por el comando.
*Marca de tiempo**	Marca de tiempo o *timestamp* (en milisegundos) de cuando el comando o evento fue grabado.
*Tamaño**	Tamaño del objeto (en bytes) para comandos *insert* o *update* .

podría ser un problema cuando un evento faltante afecta una operación posterior en la traza.

KV-replay asume que cualquier acceso (*read*, *update* or *delete*) previo a un *insert* de la clave correspondiente, indica que la clave ha sido insertada en la base de datos antes de que la traza fuera capturada. Por esta razón, existe una fase de preparación donde las claves son insertadas en la base de datos.

Sin embargo, existen otras dos explicaciones para un evento faltante: (a) La operación utiliza una clave que no existe y debería fallar, o (b) la insersión de la clave ocurrió durante el periodo de captura de la traza, pero el método de captura falló en considerarla.

El primer caso (a) puede ser inferido de la traza si esta contiene valores de retorno (ÉXITO o FRACASO); si este es el caso, el usuario puede saltar la fase de preparación y realizar una preparación manual usando un script que viene con KV-replay.

Para el segundo caso (b), el usuario puede hace la fase de preparación manual y agregar las operaciones de inserción faltantes donde el crea conveniente. Como

alternativa, se provee un script de pre-procesamiento que agrega *inserts* a claves que los necesitan, X milisegundos antes del primer acceso a la clave; siendo X un parámetro configurable.

4.3. Modelos de reproducción y garantías de ordenamiento

KV-replay soporta tres modelos de reproducción (resumidos en la tabla 4.2):

1) *Abierto (predeterminado)*: Los requerimientos son despachados de manera independientemente. Puede requerir varios clientes para trazas pesadas, es decir, si los eventos deben ser reproducidos más rápido que el máximo rendimiento (*throughput*) soportado por un cliente individual.

2) *Cerrado con ordenamiento conservativo*: Las operaciones se ejecutan solo después de que todas las operaciones previas hayan sido completadas (Zhu et al., 2005); solo está soportado para el modo con un cliente individual ejecutando un solo hilo, ya que los hilos no son sincronizados (más allá de la barrera de inicio) y el ordenamiento conservativo puede ser violado si varios hilos o clientes son utilizados.

3) *Cerrado con ordenamiento orientado a objetos*: Se garantiza que los requerimientos de un cliente sean ejecutados en el orden en que fueron grabados (Weiss et al., 2013), y después de que los requerimientos previos para dicho cliente hayan sido completados; igual que para in modelo **parcialmente abierto**.

Tabla 4.2: Modelos de reproducción y políticas soportadas por KV-replay.

Modelo de reproducción	**Garantías de ordenamiento**	**Políticas de tiempo**
Abierto	Ninguna	Original y escalada
Cerrado	Conservativa (Zhu et al., 2005)	Full, original, escalada
Cerrada/ Parcialmente-abierta	Orientada-a-recursos (Weiss et al., 2013)	Original y escalada

4.4. Políticas de tiempo

KV-replay soporta tres políticas de tiempo que determinan cuán rápido se deben reproducir los eventos.

1) *Full speed:* Los eventos son enviados uno después de otro, pero solo después de que el anterior ha sido finalizado, ya que las operaciones son invocadas utilizando llamadas con bloqueo; este es el mismo funcionamiento provisto por YCSB. Solo es aplicable en el modelo de reproducción cerrado.

2) *Velocidad original:* Los tiempos de interarribos de la traza son imitados. Si se usa con el modelo de reproducción cerrado, los eventos que están demasiado cercanos unos de otros podrían no ser reproducidos a la verlocidad original; dos eventos consecutivos con un tiempo de interarribo más corto que la latencia de la base de datos no podrían ser enviados a tiempo usando llamadas con bloqueo.

3) *Velocidad escalada:* Un factor de aceleración, f, puede escalar la velocidad de reproducción hacia arriba o hacia abajo. Incrementando gradualmente la velocidad es la manera recomendada de ejecutar una prueba de estrés con el modelo abierto.

4.5. Escalamiento de la carga de trabajo

Cuando se reproduce una traza en un sistema más pequeño o más grande que el original, se podría desear escalar la carga de trabajo aumentando o disminuyendo las dimensiones temporales, espaciales o ambas (Traeger et al., 2008). Sin embargo, se debe tomar en cuenta que el escalamiento modifica la carga de tabajo y puede afectar los resultados. Si un investigador está inclinado a escalar la carga de trabajo, las herramientas provistas hacen posible la reproducibilidad del experimento.

KV-reply soporta **escalamiento temporal** por medio de un factor de escalamiento, permitiendo incrementar la presión en el sistema o para reproducir una traza en un sistema que es de diferente tamaño a aque donde la carga de trabajo fue capturada.

KV-replay soporta tres técnicas de escalamiento espacial: Clonado (Zhu et al., 2005), sub-tracing (Zhu et al., 2005) e intensificación (Hua et al., 2014). La técnica de *clonado* escala hacia arriba (incrementa) el número de operaciones y objetos; cada traza clonada es reproducida contra un conjunto separado de claves. La técnica de *sub-tracing* realiza un escalamiento hacia abajo (decremental) mediante la descomposición de la traza en múltiples subtrazas, siendo la nueva carga de trabajo una combinación de un subconjunto de las sub-trazas obtenidas. Finalmente, el escalamiento por *intensificación* aumenta la intensidad de la carga de trabajo mediante la separación en sub-trazas que luego son alineadas en el tiempo cero. Las claves en la sub-trazas son renombradas para incluir un identificador, de tal manera que cada sub-traza acceda a un conjunto independiente de claves. Las sub-trazas combinadas son reproducidas como una carga de trabajo más pesada pero de menor duración. El factor de intensificación está dado por el número de

sub-trazas en que la traza es dividida. (Hua et al., 2014).

4.6. Limitaciones y discusión

KV-replay alcanza un niveles de precisión en milisegundos en el despacho de los eventos. Es difícil para las herramientas de *benchmarking* ejecutándose en sistemas operativos que no son de tiempo real, alcanzar niveles de precisión en el rango de los microsegundos (Anderson et al., 2004), las razones para esta limitación incluyen: Granularidad de programación y rendimiento impredecible debido a interrupciones, bloqueo, contención de recursos y complejidad del núcleo ó *kernel* del sistema operativo. Anderson et al. resolvieron el problema de alcanzar una precisión de microsegundos en un sistema operativo sin soporte de tiempo real, a través del mantenimiento de tiempo por hilado (*spinning*) y otros trucos detallados en (Anderson et al., 2004). En el futuro, estas técnicas pueden ser incorporadas en KV-replay. Sin embargo, las pruebas realizadas muestran que estos errores afectans solo los tiempos obtenidos vs. los tiempos esperados por evento. Otras dimensiones temporales, como los interarribos y la variabilidad en la intensidad de los pedidos (*busrtiness*) son reproducidas con exactitud (ver Capítulo 5).

Las evaluaciones mediante reproducción de trazas también son afectadas por los métodos que se utilizaron para capturar las trazas (Traeger et al., 2008), Sin embargo, las limitaciones o problemas relacionados con los métodos de captura están fuera del alcance de este trabajo.

Finalmente, se requiere trazas de sistemas de almacenamiento de tipo clave-valor disponibles públicamente. Hasta que dichas trazas estén disponibles, otras trazas pueden ser utilizadas, como aquellas correspondientes a accesos a videos de

YoutTube (Zink et al., 2009), descargas de la Wikipedia (Urdaneta et al., 2009) y requerimientos del sistema de archivos HDFS de Yahoo (Yahoo Webscope).

Capítulo 5

Evaluación experimental

La evaluación de KV-replay está enfocada en medir y comprobar su exactitud, alto rendimiento y utilidad. Los datos mostrados para cada experimento corresponden a los valores promedios obtenidos durante cinco ejecuciones. Los experimentos que comparan los resultados de YCSB versus KV-replay utilizan un modelo de reproducción cerrado a máxima velocidad, de tal manera que la comparación sea justa, ya que es el modelo utilizado por YCSB.

5.1. Configuración experimental

Los experimentos fueron ejecutados en máquinas virtuales corriendo el sistema operativo CentOS v7.2.1511 x86_64, con 1GB de memoria RAM y 1 vCPU, utilizando VirtualBox v5.1.2 como plataforma de virtualización.

Los servidores anfitriones utilizan la misma versión de CentOS y cuentan con 4GB de memoria RAM y procesadores Intel Core2 Quad CPU Q6600 @ 2.40GHz.

Las versiones de las bases de datos utilizadas fueron: Redis 2.8.19, Redis Cluster 3.2.3, Memcached 1.4.15, Riak KV 2.1.4 y Hazelcast 3.6.4.

5.2. Cargas de trabajo

Se utilizó una traza de peticiones a videos de YouTube (Zink et al., 2009), específicamente la traza T5 que contiene peticiones de 16K clientes hacia 300K videoclips. Los videos son servidos por redes de entrega de contenidos[1], pero la carga de trabajo está fuertemente correlacionada con un caso de uso común de sistemas de almacenamiento de clave-valor: Contadores de interacción con el contenido (visualizaciones, likes, dislikes, etc). Estos contadores son recuperados al mismo tiempo que el video es solicitado, ya que en 2008 todos los videos de YouTube eran reproducidos de manera automática cuando la página del video era cargada[2].

Cada registro en la traza representa un pedido individual y contiene los siguientes campos: Estampa de tiempo, dirección IP del servidor de YouTube, dirección IP del cliente, tipo de pedido, identificador del video y dirección IP del servidor de contenido. Una etapa de pre-procesamiento fue utilizada para convertir la traza al formato requerido por KV-replay.

Esta carga de trabajo fue utilizada de dos maneras:

1. La traza fue reproducida utilizando KV-replay.

2. Se encontró el mejor ajuste a una distribución Zipf para la distribución de popularidad de los objetos de YouTube (coeficiente = 0,5, Figura 5.1) y se configuró YCSB para que la distribución Zipfian utilice este parámetro. Para encontrar el mejor ajuste a los datos, se utilizó el método de descartar la cola de la distribución (Motwani y Vassilvitskii, 2006) y se aplicó el algoritmo

[1] CDN por sus siglas en inglés.
[2] Los videos aún son reproducidos de manera automática, pero solo por la pestaña del navegador que está en primer plano.

descrito en (Zemlys, 2011).

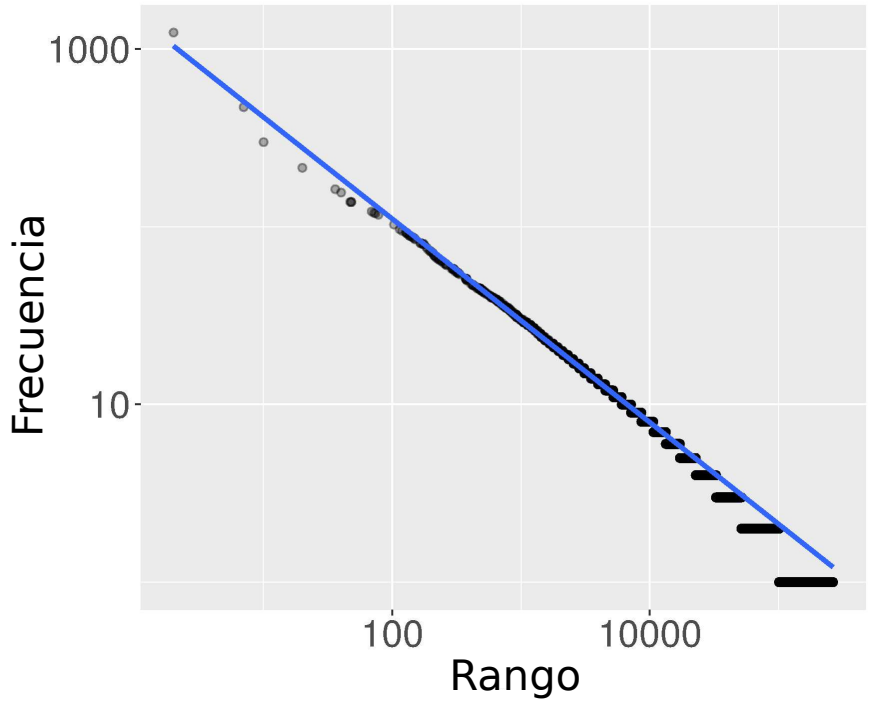

Figura 5.1: Distribución de Rango-frecuencia de la popularidad de los videos en la traza de YouTube. La línea muestra el mejor ajuste a la distribución Zipfian. Cada punto representa un objeto único. Aunque se utilizó el conjunto de datos completos para realizar el ajuste a la curva, solo se grafica una muestra aleatoria del 10 % de los objetos.

Para facilitar la comparación de resultados, los tamaños de las claves y valores fueron iguales en todos los experimentos (23 y 1K bytes, respectivamente).

YCSB soporta otras distribuciones de popularidad adicionales a Zipf, por lo que se ejecutaron experimentos utilizando las distribuciones Zipf, Uniforme y Latest. Sin embargo, se reportaron solo los resultados de Zipf debido a que fue la que obtuvo las mejores aproximaciones a los resultados obtenidos con la traza real (menores errores cuadrados de la raíz media, RMSE), de esta manera se obtuvo la comparación más justa. Adicionalmente, la distribución Zipf de YCSB fue la distribución más comúnmente utilizada en la bibliografía revisada.

5.3. Bases de datos evaluadas

Los almacenes de datos de tipo clave-valor en memoria están diseñados para ser muy rápidos, lo que los convierte en un objeto de estudio frecuente (Zhang et al., 2014). Los experimentos para la evaluación de KV-replay se realizaron con los cuatro principales sistemas de almacenamiento de tipo clave-valor (Db-engines.com, 2017): Redis, Memcached, Riak KV (basado en la arquitectura de DynamoDB) y Hazelcast, con sus respectivas configuraciones por defecto, excepto por Riak KV, donde se utilizaron 8 particiones en lugar de 64 para obtener una curva de tasa de fallos más suavizada, similar a las obtenidas con las otras bases de datos.

Se utilizó una de las aplicaciones más comunes de los almacenes de datos de tipo clave-valor en memoria: las memorias caché. Con este objetivo se configuró las mencionadas bases de datos para que utilicen la política de desalojo LRU (Least Recently Used) y se utilizaron varios tamaños de la memoria caché.

5.4. Precisión

Para medir la precisión en el despacho de eventos de KV-replay, se ejecutaron pruebas utilizando tanto una instancia individual del reproductor, como una prueba distribuida con la carga de trabajo dividida entre diez clientes. Se utilizó el modelo de reproducción abierto con la política de tiempo "original". La meta de experimento era comprobar si KV-replay podría reproducir tres métricas relacionadas con el tiempo: el tiempo de envío de los eventos, los tiempos de interarribos y variabilidad o presencia de picos con mayor intensidad (*burstiness*) en los pedidos. Se reportan los resultados de la prueba distribuida, considerando

que los resultados del cliente individual fueron ligeramente mejores.

La Figura 5.2 muestra la precisión alcanzada por KV-replay al reproducir los tiempos por evento almacenados en la traza con un error pequeño (mediana $= 2{,}04ms$).

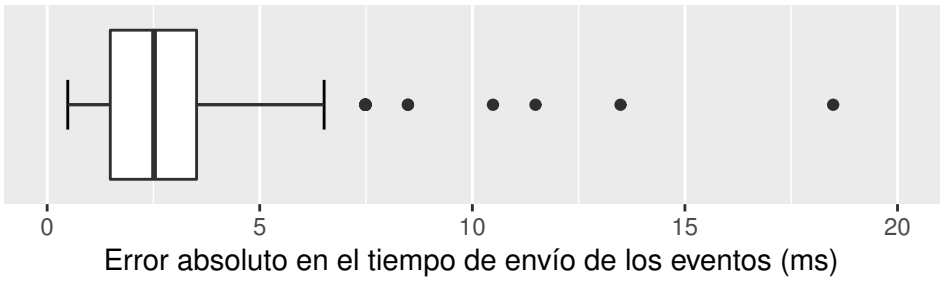

Figura 5.2: Precisión of KV-replay: Diagrama de cajas de los errores absolutos en los tiempos de despacho de eventos, obtenidos con la carga de trabajo reproducida por KV-replay.

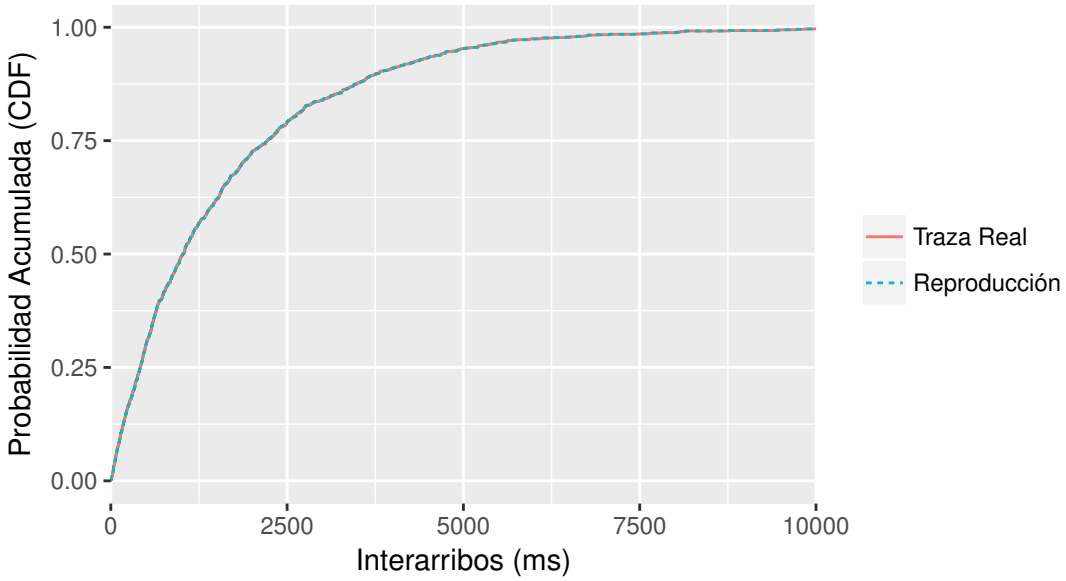

Figura 5.3: Precisión de KV-replay: Funciones de probabilidad acumulada (CDF) de los tiempos de interarribo para la traza real y la carga de trabajo reproducida por KV-replay. Las curvas superpuestas grafican la exactitud de la reproducción de la carga. Para mejor apreciación, se muestran únicamente los interarribos de hasta 10000 ms.

En la Figura 5.3 se observan que las funciones de distribución acumuladas (CDFs) de los interarribos de la traza real vs. los de la carga reproducida son casi idénticas. La distancia de Kolmogorov-Smirnov entre las dos curvas es de 0,0045, lo que significa un error menor a 0,46 puntos porcentuales.

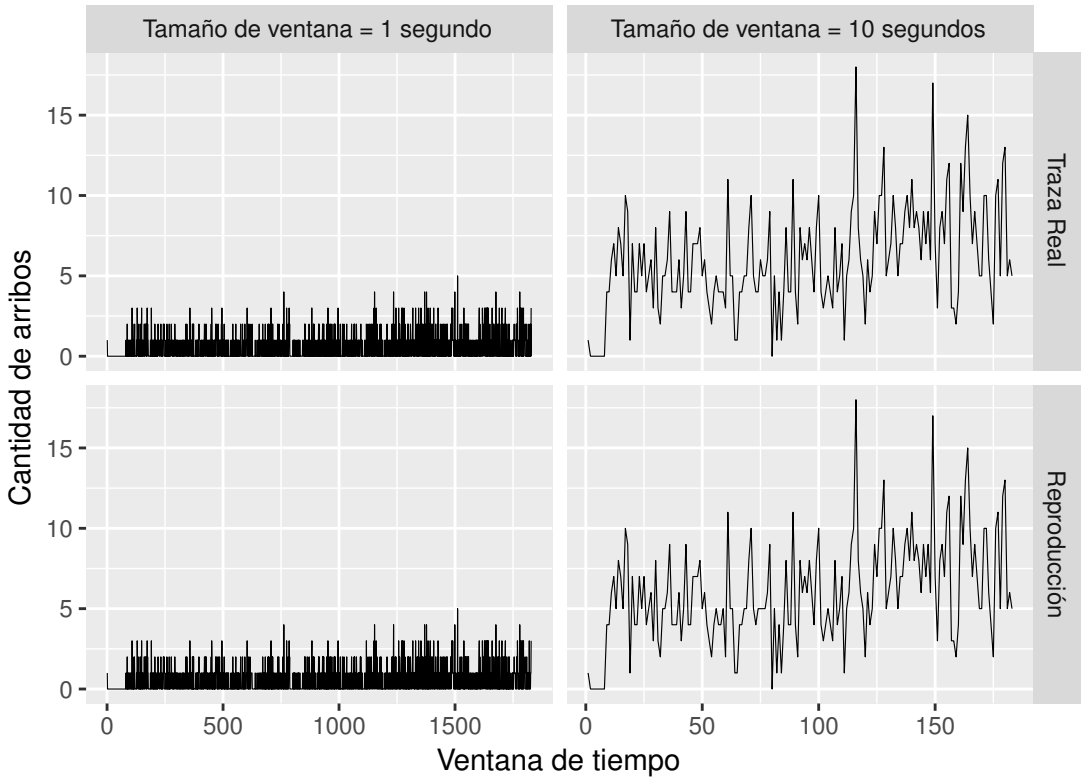

Figura 5.4: Precisión de KV-replay: Variabilidad o fluctuaciones en la intensidad de los pedidos. Se observa que las fluctuaciones son similares en diferentes escalas (longitud de ventana de tiempo).

La variabiliad en la intensidad de los pedidos (*burstiness*) también es reproducida con exactitud, tal como se observa en la Figura 5.4. Esta similitud se cuantifica calculando el exponente de Hurst de las series de tiempo de los arribos, utilizando un análisis R/S y una ventana de 1K milisegundos (0.7823 vs. 0.7818, para la traza real y la reproducción respectivamente). El parámetro de Hurst es una medida de fluctuación de intensidades de una serie de tiempo, que en este

caso corresponde al proceso de conteo de los arribos en una ventana de tiempo predefinida (Crovella y Bestavros, 1997).

Se debe notar que parte del error en las medidas de tiempo es debido a la sincronización de los relojes entre las máquinas virtuales. Cuando se ejecutan todos los diez clientes en una misma máquina, los tiempos son más exactos: Mediana de 1,36ms en lugar de 2,04ms y percentil 90 % de 3,6ms en lugar de 5,5ms).

5.5. Utilidad

La meta de este conjunto de experimentos fue determinar cómo una dimensión de la carga de trabajo, no reproducida por YCSB, afecta los resultados de la evaluación de rendimiento: Localidad temporal realista. Se ejecutaron experimentos utilizando YCSB (Zipf) y KV-replay, ambos accediendo a aproximadamente el mismo número de claves[3].

La Figura 5.5 muestra que los errores entre las curvas de tasas de fallos (MRCs) de las cargas de trabajo sintéticas versus las reales, pueden ser tan hasta del 30 %. La carga de trabajo sintética no aproxima los resultados reales debido a la falta de localidad temporal[4]. Como resultado, un sistema de caché puede ser provisionado incorrectamente teniendo un impacto negativo en el costo y rendimiento.

La Figura 5.6 compara las curvas de tasas de fallos (MRCs) de las políticas de desalojo de diferentes bases de datos, ya que éstas raramente implementan LRU;

[3]No es posible indicar a YCSB cuántas claves exactas utilizar durante una ejecución, pero se le puede indicar el número de claves que se debe insertar en la fase de carga; a través de un procedimiento de prueba y error, se pudo determinar que insertando 600K registros se podía aproximar el número de claves presentes en la traza fe YouTube, con una diferencia dentro del 1,3 %. Dado que IRM utiliza un muestreo aleatorio del espacio de claves, resulta imposible acceder a un número exacto de claves en cada ejecución.

[4]Utilizando YCSB con la distribución *Latest* se obtuvo resultados aún peores debido a que simula una carga de trabajo con una localidad temporal significativamente más alta que la de la traza de YouTube.

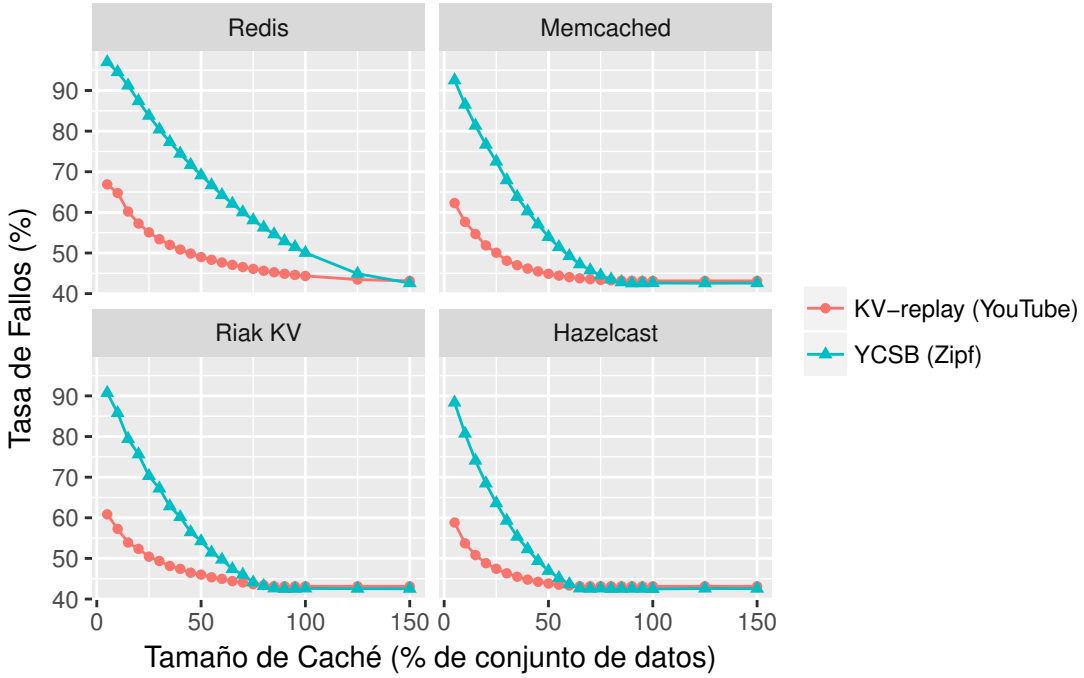

Figura 5.5: Curvas de Tasa de Fallos (*MRCs*) para bases de datos de tipo clave-valor en memoria, actuando como cachés con política de desalojo LRU, para dos cargas de trabajo: Zipf (YCSB) y una traza real (KV-replay). Las cargas sintéticas (Zipf) carecen de localidad temporal, por lo que sus resultados obtienen baja precisión.

por el contrario utilizan alguna aproximación (e.g., para reducir la contención de bloqueo). El *benchmark* sintético sobrestima considerablemente las diferencias entre las MRCs.

La Figura 5.7 muestra la aceleración (*speedup*), en latencia de operaciones de lectura, que se obtiene de utilizar una caché que puede almacenar todas las claves versus una caché pequeña que solo puede mantener el 10 % de los datos. Los resultados muestran que el *benchmark* sintético sobrestima la aceleración hasta en un 18 %.

El tamaño de los registros también afecta los resultados. Para comprobar esto, se utilizó Memcached, que utiliza *particiones (slabs)* para almacenar objetos

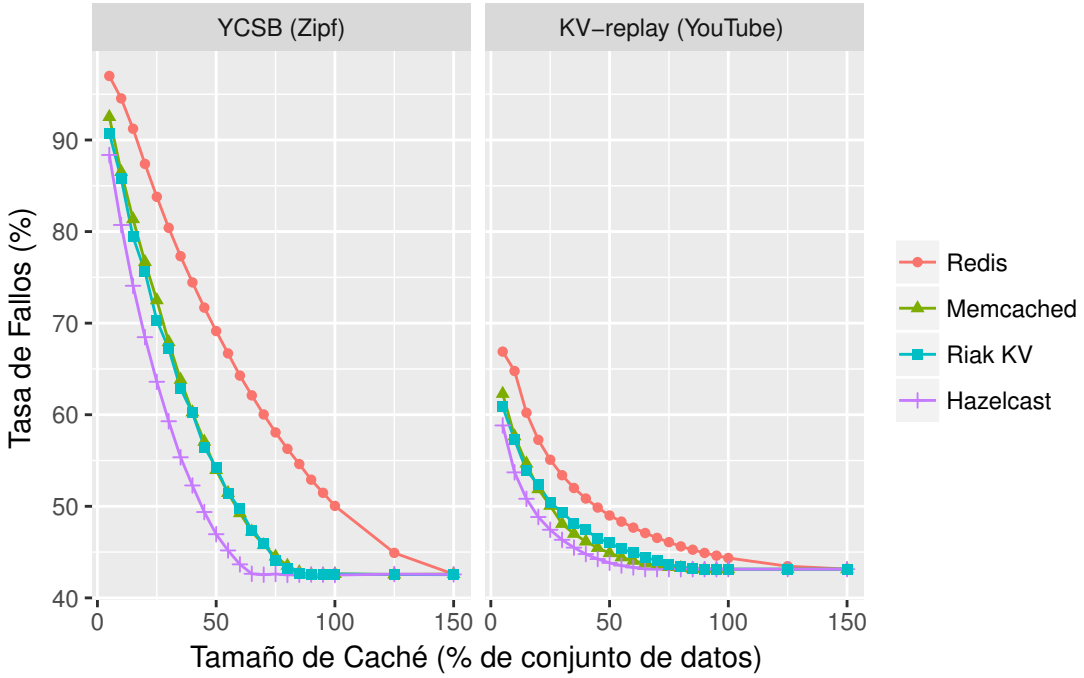

Figura 5.6: Curvas de Tasa de Fallos (*MRCs*) para bases de datos de tipo clave-valor en memoria, utilizando cargas de trabajo sintéticas y reales. La diferencia entre las curvas para la misma carga de trabajo está dada por las diferentes implementaciones de las políticas de desalojo (LRU en este caso). La evaluación sintética (YCSB) sobrestima la diferencia entre las curvas.

de diferentes rangos de tamaño. Si todos los valores tienen el mismo tamaño, la contención en la partición correspondiente se incrementa, resultando en un incremento en la latencia y una disminución del rendimiento (*throughput*), como se muestra en la Figura 5.8. Se configuró Memcached con tamaños de caché variables y se obtuvo resultados similares en todos los casos; la figura muestra los resultados para un tamaño de caché de 1.5GB. Para generar la carga de trabajo, se utilizó KV-replay con la traza de YouTube de dos maneras: Con un tamaño de valor fijo (igual que en el funcionamiento por defecto de YCSB) y con registros de tamaño variable. Para lo segundo, los tamaños de los valores fueron elegidos de tal manera que los registros fueran distribuidos uniformemente a través de todas

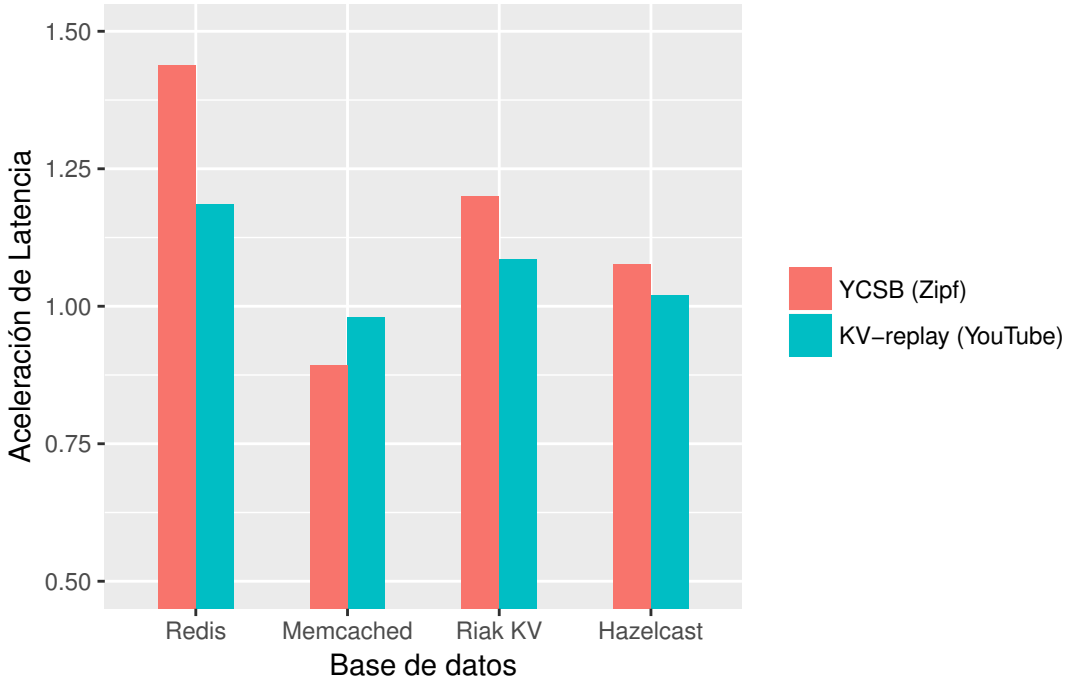

Figura 5.7: Aceleración (en latencia de lecturas) obtenida de utilizar una caché grande que puede almacenar todo el conjunto de datos, versus una caché que solo puede almacenar el 10 % de los datos.

las particiones de Memcached; para el primer caso, el tamaño de los valores fue determinado como el tamaño medio de los valores utilizados en el experimento de tamaño variable (93862 bytes).

También se consideró un experimento de balanceo de carga en un clúster de 10 nodos con Redis. El clúster de Redis divide el espacio de claves entre los nodos disponibles. Se ejecutaron experimentos a velocidad máxima y se registró el número de accesos por nodo para cada ventana de tiempo de 1 segundo. La Figura 5.9 muestra el gráfico de cajas del número de accesos por ventana de tiempo; las diferencias en el rango ($max - min$) y el rango intercuartil ($IQR = Q3 - Q1$) muestran que los resultados para la carga de trabajo sintética son más dispersos. Debido a la localidad temporal presente en la carga de trabajo de

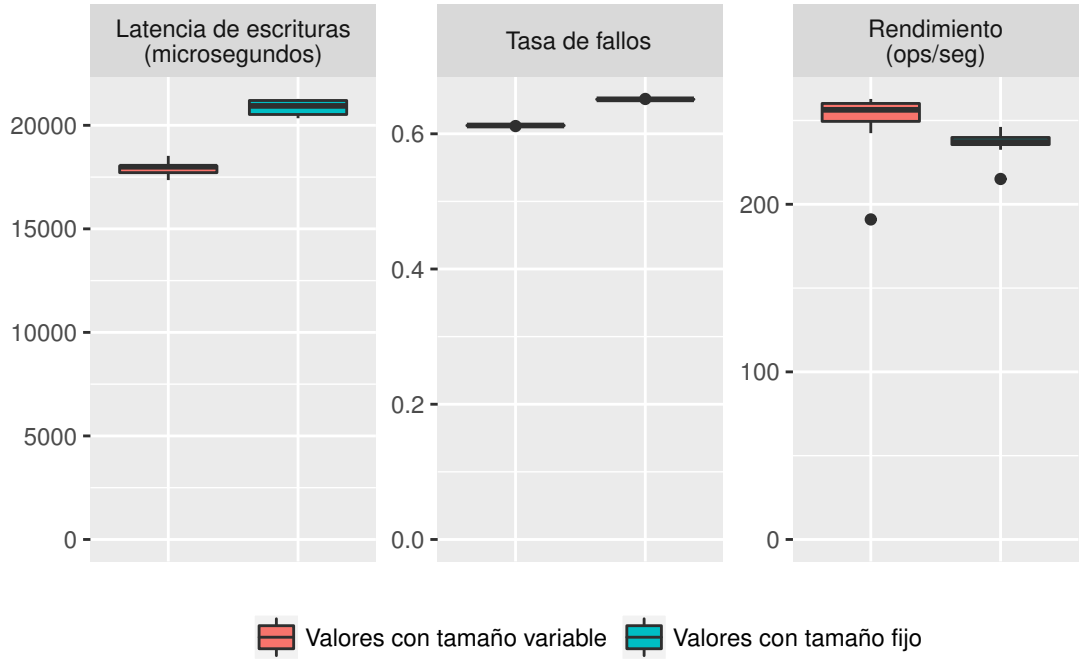

Figura 5.8: Operación de Memcached con inserciones de registros de tamaño fijo, versus registros de tamaño variable. Al utilizar registros de tamaño variable (KV-replay) disminuyen tanto la latencia de las operaciones de escritura como la tasa de fallos, mientras que el aumenta el número de operaciones por segundo.

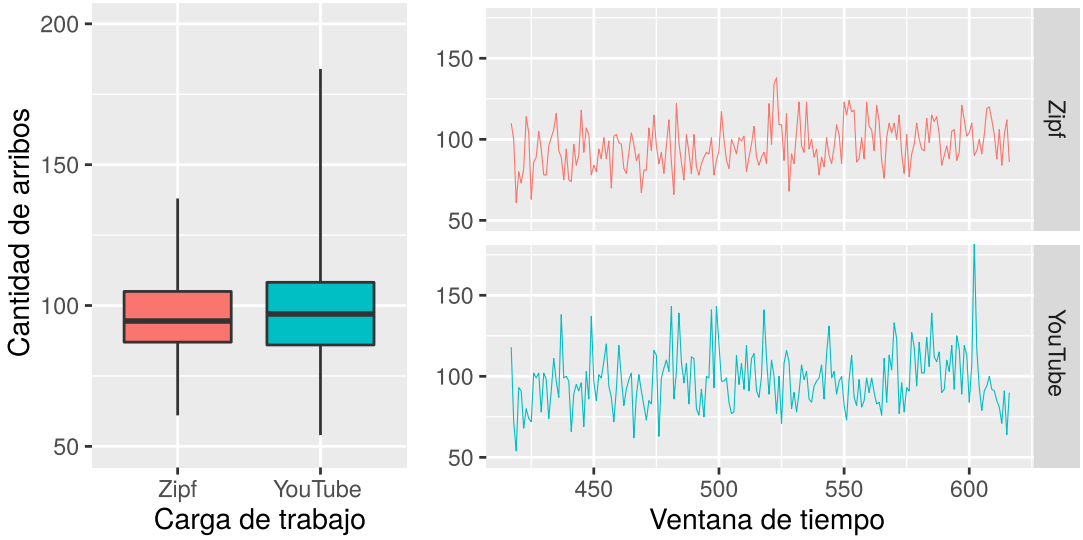

Figura 5.9: Hotspots de accesos en un Cluster Redis de 10 nodos. A la izquierda: Gráfico de cajas con el valor agregado del número de pedidos que cada nodo recibió en cada ventana de 1 segundo. A la derecha: Una muestra de los arribos, para uno de los nodos en cada experimento.

YouTube, los accesos a una clave tienden a ser cercanos en el tiempo, creando *hotspots*, mientras que en la carga de trabajo sintética, generada a partir de una distribución Zipfian, los accesos a la misma clave están distribuidos aleatoriamente a través de la duración del experimento (algunos *hotspots* aparece, pero debido a claves con una alta popularidad, no debido a la localidad temporal)

5.6. Rendimiento

El objetivo de estos experimentos fue observar si KV-replay podría despachar eventos tán rápido como YCSB. Para KV-replay, se utilizó el modelo de reproducción cerrado con múltiples hilos, el cual puede violar el ordenamiento conservativo (Zhu et al., 2005). En este experimento se asume que los eventos son independientes, pero se utiliza el modelo de reproducción cerrado porque en éste

se reduce el rendimiento por el uso de llamadas con bloqueo. Tampoco se utilizó otros modelos de reproducción debido a que no son soportados por YCSB y por lo tanto los resultados no serían comparables.

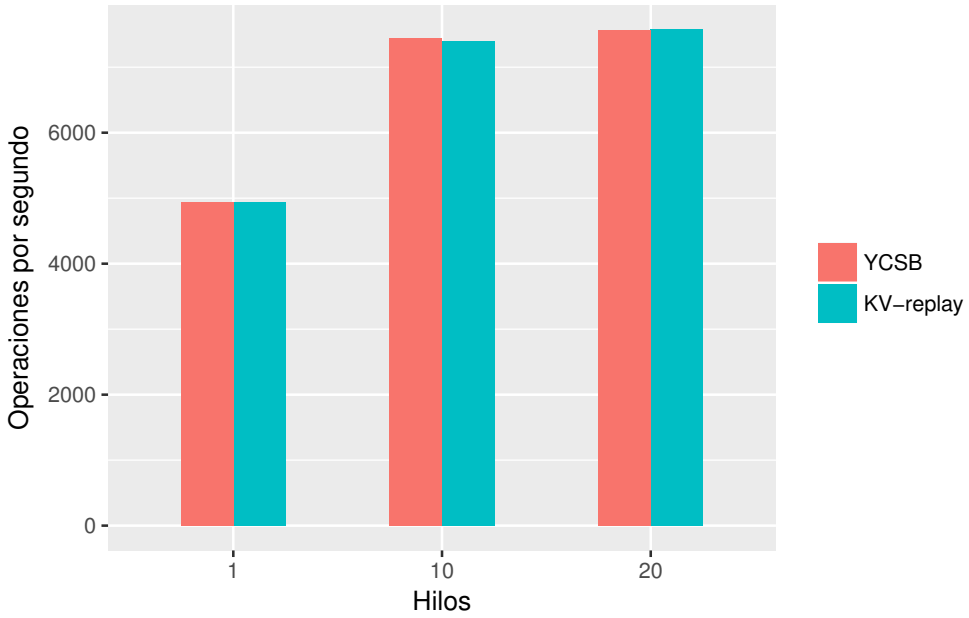

Figura 5.10: Operaciones por segundo generadas por YCSB y KV-replay, con 1, 10 y 20 hilos. A máxima velocidad, KV-replay puede generar pedidos tan rápido como YCSB.

En los resultados, mostrados en la Figura 5.10, se observa que no existe diferencia significativa en el rendimiento, lo que indica que KV-replay puede estresar el sistema tan efectivamente como YCSB.

Capítulo 6

Conclusiones y recomendaciones

En el presente trabajo se describe un modelo para evaluar sistemas de almacenamiento de clave-valor en la nube, mediante la reproducción de cargas de trabajo reales a partir de trazas almacenadas en archivos de texto. Este modelo ha sido implementado en una herramienta de código abierto llamada KV-replay, que está disponible para descarga en https://github.com/ebozag/KV-replay.

KV-replay es un reproductor de trazas basado en eventos, que utiliza un modelo de reproducción abierta y soporta tres políticas de tiempo para el despacho de eventos: máxima velocidad, velocidad original y velocidad escalada.

Adicionalmente, KV-replay también soporta el modelo de reproducción cerrado con una política de tiempo conservativa, pero solo aplicable para el caso de único-cliente o único hilo.

Los resultados experimentales demuestran que KV-replay es tan rápido como YCSB, exacto en la reproducción de interarribos y de fluctuaciones en intensidad de los arribos, y útil, ya que permite evaluar sistemas de almacenamiento de clave-valor con cargas de trabajo reales.

Se muestra que KV-replay puede ayudar a evaluar de mejor manera el rendimiento de sistemas, ya que utilizando cargas de trabajo sintéticas se obtienen errores significativos en la evaluación, tales como un 33 % en la curva de tasa de fallos (miss rate curve) para tamaños de caché pequeños; y una sobreestimación del 18 % en la aceleración del tiempo de respuesta que obtiene el sistema al incrementar el tamaño de la caché.

Bibliografía

Cristina Abad, Mindi Yuan, Chris Cai, Yi Lu, Nathan Roberts, y Roy Campbell. Generating request streams on Big Data using clustered renewal processes. *Performance Evaluation*, 70(10), 2013.

Ahmad Al-Shishtawy y Vladimir Vlassov. Elastman: Elasticity manager for elastic key-value stores in the cloud. En *ACM Cloud and Autonomic Computing Conference (CAC)*. 2013.

Eric Anderson, Mahesh Kallahalla, Mustafa Uysal, y Ram Swaminathan. Buttress: A toolkit for flexible and high fidelity I/O benchmarking. En *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*. 2004.

Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, y Mike Paleczny. Workload analysis of a large-scale key-value store. En *ACM SIGMETRICS Performance Evaluation Review*, tomo 40. 2012.

Giuseppe Bianchi, Andrea Detti, Alberto Caponi, y Nicola Blefari Melazzi. Check before storing: What is the performance price of content integrity verification in LRU caching? *SIGCOMM Computer Communication Review*, 43(3), 2013.

Y. Chen, A. Ganapathi, R. Griffith, y R. Katz. The case for evaluating MapReduce performance using workload suites. En *Proceedings of the IEEE International*

Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). 2011.

Brian Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, y Russell Sears. Benchmarking cloud serving systems with YCSB. En *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*. 2010.

Mark E. Crovella y Azer Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6), 1997.

Db-engines.com, 2017. DB-engines ranking of key-value stores [online]. http://db-engines.com/en/ranking/key-value+store, 2016. [Fecha de Acceso: 18-Julio-2016].

Eric Evans. Nosql: What's in a name? 2009. URL http://blog.sym-link.com/2009/10/30/nosql_whats_in_a_name.html. [Fecha de Acceso: 21-Agosto-2017].

Marc Gravell. Blogpost: async redis await booksleeve [online]. http://blog.marcgravell.com/2011/04/async-redis-await-booksleeve.html, 2011. [Fecha de Acceso: 21-Agosto-2017].

Abdul Haseeb y Geeta Pattun. A review on nosql: Applications and challenges. *International Journal of Advanced Research in Computer Science*, 8(1), 2017.

Y. Hua, H. Jiang, Y. Zhu, D. Feng, y L. Xu. SANE: Semantic-aware namespace in ultra-large-scale file systems. *IEEE Transactions on Parallel and Distributed Systems*, 25(5), 2014.

Nikolai Joukov, Timothy Wong, y Erez Zadok. Accurate and efficient replaying of file system traces. En *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*. 2005.

Michael P. Mesnier, Matthew Wachs, Raja R. Sambasivan, Julio Lopez, James Hendricks, Gregory R. Ganger, y David O'Hallaron. TRACE: Parallel trace replay with approximate causal events. En *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*. 2007.

Rajeev Motwani y Sergei Vassilvitskii. Distinct values estimators for power law distributions. En *SIAM ANALCO*. 2006.

Thiago Emmanuel Pereira, Francisco Brasileiro, y Lívia Sampaio. File system trace replay methods through the lens of metrology. En *Proceedings of the IEEE Symposium on Mass Storage Systems and Technologies (MSST)*. 2016.

Rekha Pitchumani, Shayna Frank, y Ethan L. Miller. Realistic request arrival generation in storage benchmarks. En *Proceedings of the IEEE Symposium on Mass Storage Systems and Technologies (MSST)*. 2015.

Meikel Poess y Chris Floyd. New TPC benchmarks for decision support and web commerce. *SIGMOD Rec.*, 29(4), 2000.

Bianca Schroeder, Adam Wierman, y Mor Harchol-Balter. Open versus closed: A cautionary tale. En *Proceedings of the USENIX Conference on Networked Systems Design & Implementation (NSDI)*. 2006.

Stackoverflow.com, 2017. Stackoverflow question: pub/sub middleware with persistent storage [online]. http://stackoverflow.com/questions/19117153, 2013. [Fecha de Acceso: 21-Agosto-2017].

Carlo Strozzi. Nosql-a relational database management system. *Lainattu*, 5:2014, 1998.

Vasily Tarasov, Santhosh Kumar, Jack Ma, Dean Hildebrand, Anna Povzner, Geoff Kuenning, y Erez Zadok. Extracting flexible, replayable models from large block traces. En *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*. 2012.

A. Traeger, E. Zadok, N. Joukov, y C. Wright. A nine year study of file system and storage benchmarking. *ACM Transactions on Storage (TOS)*, 4(2), 2008.

Stefano Traverso, Mohamed Ahmed, Michele Garetto, Paolo Giaccone, Emilio Leonardi, y Saverio Niccolini. Temporal locality in today's content caching: Why it matters and how to model it. *SIGCOMM Computer Communication Review*, 43(5), 2013.

Guido Urdaneta, Guillaume Pierre, y Maarten van Steen. Wikipedia workload analysis for decentralized hosting. *Computer Networks*, 53(11), 2009.

Sarut Vanichpun y Armand M. Makowski. Comparing strength of locality of reference – Popularity, majorization, and some folk theorems. En *Proceedings of the IEEE International Conference on Computing Communications (INFOCOM)*. 2004a.

Sarut Vanichpun y Armand M. Makowski. The output of a cache under the Independent Reference Model: Where did the locality of reference go? *SIGMETRICS Performance Evalualtion Review*, 32(1), 2004b.

Zev Weiss, Tyler Harter, Andrea Arpaci-Dusseau, y Remzi Arpaci-Dusseau. ROOT: Replaying multithreaded traces with resource-oriented ordering. En

Proceedings of the ACM Symposium on Operating Systems Principles (SOSP). 2013.

Yahoo Webscope. Yahoo Hadoop grid logs, version 1.0. https://webscope.sandbox.yahoo.com. [Fecha de Acceso: 21-Agosto-2017].

Vaidotas Zemlys. Answer to: How to calculate zipf's law coefficient from a set of top frequencies? [online]. http://stats.stackexchange.com/questions/6780/how-to-calculate-zipfs-law-coefficient-from-a-set-of-top-frequencies, 2011. [Fecha de Acceso: 21-Agosto-2017].

Hao Zhang, Bogdan Marius Tudor, Gang Chen, y Beng Chin Ooi. Efficient in-memory data management: An analysis. *Proceedings of the VLDB Endowment*, 7(10), 2014.

Ningning Zhu, Jiawu Chen, y Tzi Chiueh. TBBT: Scalable and accurate trace replay for file server evaluation. En *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*. 2005.

Michael Zink, Kyoungwon Suh, Yu Gu, y Jim Kurose. Characteristics of YouTube network traffic at a campus network - measurements, models, and implications. *Computer Networs*, 53(4), 2009.