Informe:

Sin compromisos: transacciones distribuidas con

consistencia, disponibilidad y rendimiento

Autores Originales: Aleksandar Dragojevi´c, Dushyanth Narayanan, Edmund B. Nightingale,

Matthew Renzelmann, Alex Shamis, Anirudh Badam, Miguel Castro

**Resumen**

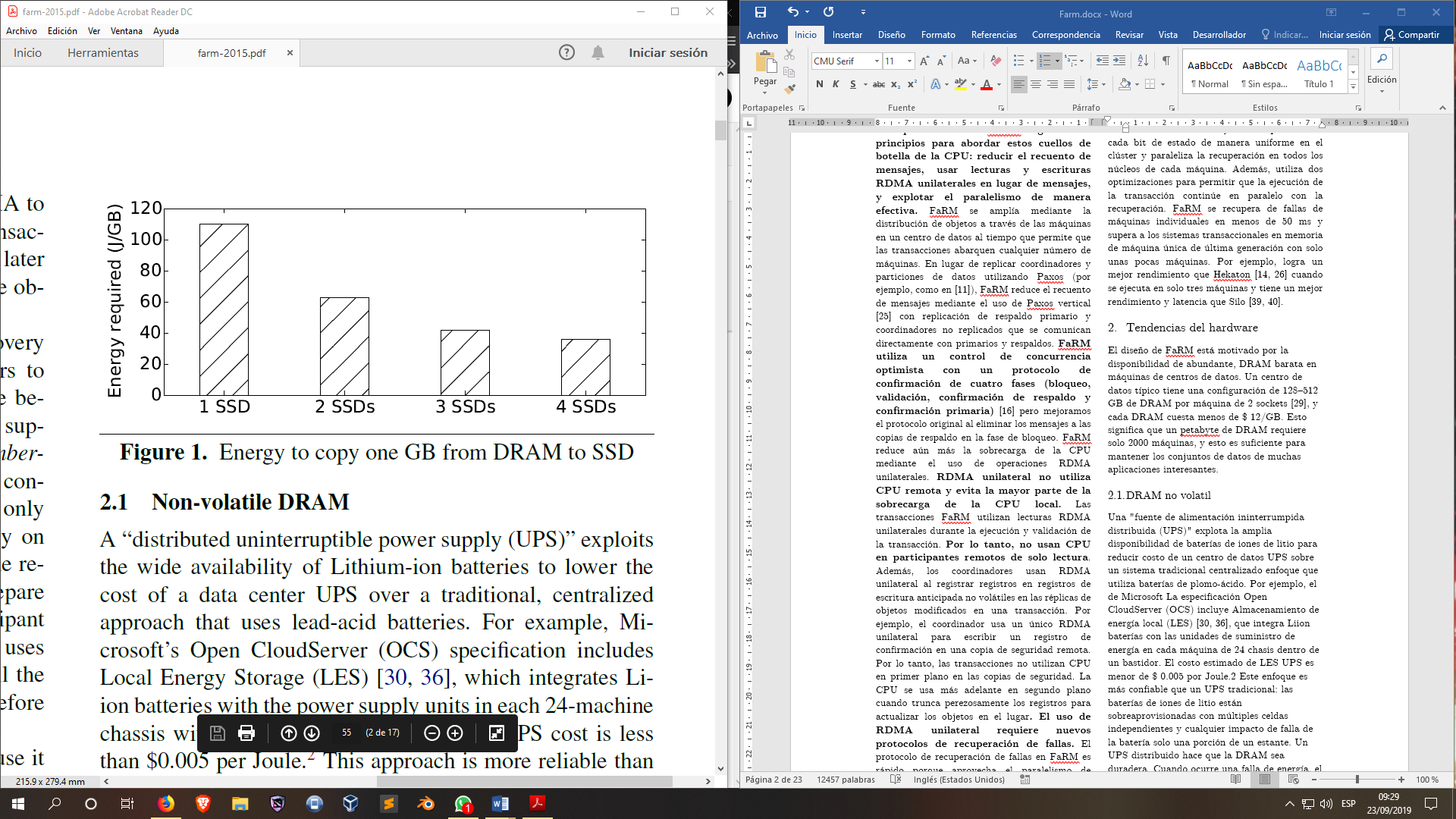
Las transacciones (Una transacción generalmente representa cualquier cambio en una base de datos) con alta consistencia y disponibilidad simplifican la construcción y el razonamiento sobre sistemas distribuidos. Sin embargo, las implementaciones anteriores funcionaron mal. Esto obligó a los diseñadores de sistemas a evitar transacciones por completo, a debilitar las garantías de coherencia o a proporcionar transacciones de máquinas individuales que requieren que los programadores particionen sus datos. En este documento, se muestra que no hay necesidad de comprometerse en los centros de datos modernos. Mostramos que una plataforma de computación distribuida de memoria principal llamada FaRM puede proporcionar transacciones distribuidas con serialización estricta, alto rendimiento, durabilidad y alta disponibilidad. FaRM logra un rendimiento máximo de 140 millones de transacciones TATP (es una prueba diseñada para medir el rendimiento de los sistemas de transacciones de bases de datos en memoria) por segundo en 90 máquinas con una base de datos de 4.9 TB, y se recupera de una falla en menos de 50 ms. La clave para lograr estos resultados fue el diseño de nuevos protocolos de transacción, replicación y recuperación a partir de los primeros principios para aprovechar las redes de productos básicos con RDMA y un nuevo enfoque económico para proporcionar DRAM no volátil.

1. **Introducción**

Las transacciones con alta disponibilidad y serialización estricta [35] simplifican la programación y el razonamiento sobre los sistemas distribuidos al proporcionar una abstracción simple y poderosa: una sola máquina que nunca falla y que ejecuta una transacción a la vez en un orden consistente en tiempo real. Sin embargo, los intentos anteriores de implementar esta abstracción en un sistema distribuido resultaron en un bajo rendimiento. Por otro lado, los sistemas como Dynamo [13] o Memcached [1] mejoraron el rendimiento al no admitir transacciones o al implementar garantías de consistencia débil. Otros (p. Ej., [3–6, 9, 28]) proporcionan transacciones solo cuando todos los datos residen en una sola máquina, lo que obliga a los programadores a dividir sus datos y complica el razonamiento sobre la corrección. Este documento demuestra que el nuevo software en los centros de datos modernos puede eliminar la necesidad de comprometer. Describimos los protocolos de transacción, replicación y recuperación en FaRM [16], una plataforma de computación distribuida de memoria principal. **FaRM proporciona transacciones ACID (Atómicas, consistentes, aisladas y duraderas) distribuidas con serialización estricta, alta disponibilidad, alto rendimiento y baja latencia.** Estos protocolos se diseñaron a partir de los primeros principios para aprovechar dos tendencias de hardware que aparecen en los centros de datos: redes rápidas de productos básicos con RDMA (acceso directo a memoria remota) y un enfoque económico para proporcionar DRAM no volátil. La no volatilidad se logra conectando las baterías a las unidades de fuente de alimentación y escribiendo los contenidos de DRAM en la SSD cuando falla la alimentación. Estas tendencias eliminan los cuellos de botella del almacenamiento y la red, pero también exponen los cuellos de botella de la CPU que limitan su beneficio de rendimiento. **Los protocolos de FaRM siguen tres principios para abordar estos cuellos de botella de la CPU: reducir el recuento de mensajes, usar lecturas y escrituras RDMA unilaterales en lugar de mensajes, y explotar el paralelismo de manera efectiva.** FaRM se amplía mediante la distribución de objetos a través de las máquinas en un centro de datos al tiempo que permite que las transacciones abarquen cualquier número de máquinas. En lugar de replicar coordinadores y particiones de datos utilizando Paxos (por ejemplo, como en [11]), FaRM reduce el recuento de mensajes mediante el uso de Paxos vertical [25] con replicación de respaldo primario y coordinadores no replicados que se comunican directamente con primarios y respaldos. **FaRM utiliza un control de concurrencia optimista con un protocolo de confirmación de cuatro fases (bloqueo, validación, confirmación de respaldo y confirmación primaria)** [16] pero mejoramos el protocolo original al eliminar los mensajes a las copias de respaldo en la fase de bloqueo. FaRM reduce aún más la sobrecarga de la CPU mediante el uso de operaciones RDMA unilaterales. **RDMA unilateral no utiliza CPU remota y evita la mayor parte de la sobrecarga de la CPU local.** Las transacciones FaRM utilizan lecturas RDMA unilaterales durante la ejecución y validación de la transacción. **Por lo tanto, no usan CPU en participantes remotos de solo lectura**. Además, los coordinadores usan RDMA unilateral al registrar registros en registros de escritura anticipada no volátiles en las réplicas de objetos modificados en una transacción. Por ejemplo, el coordinador usa un único RDMA unilateral para escribir un registro de confirmación en una copia de seguridad remota. Por lo tanto, las transacciones no utilizan CPU en primer plano en las copias de seguridad. La CPU se usa más adelante en segundo plano cuando trunca perezosamente los registros para actualizar los objetos en el lugar**. El uso de RDMA unilateral requiere nuevos protocolos de recuperación de fallas.** El protocolo de recuperación de fallas en FaRM es rápido porque aprovecha el paralelismo de manera efectiva. Distribuye la recuperación de cada bit de estado de manera uniforme en el clúster y paraleliza la recuperación en todos los núcleos de cada máquina. Además, utiliza dos optimizaciones para permitir que la ejecución de la transacción continúe en paralelo con la recuperación. FaRM se recupera de fallas de máquinas individuales en menos de 50 ms y supera a los sistemas transaccionales en memoria de máquina única de última generación con solo unas pocas máquinas. Por ejemplo, logra un mejor rendimiento que Hekaton [14, 26] cuando se ejecuta en solo tres máquinas y tiene un mejor rendimiento y latencia que Silo [39, 40].

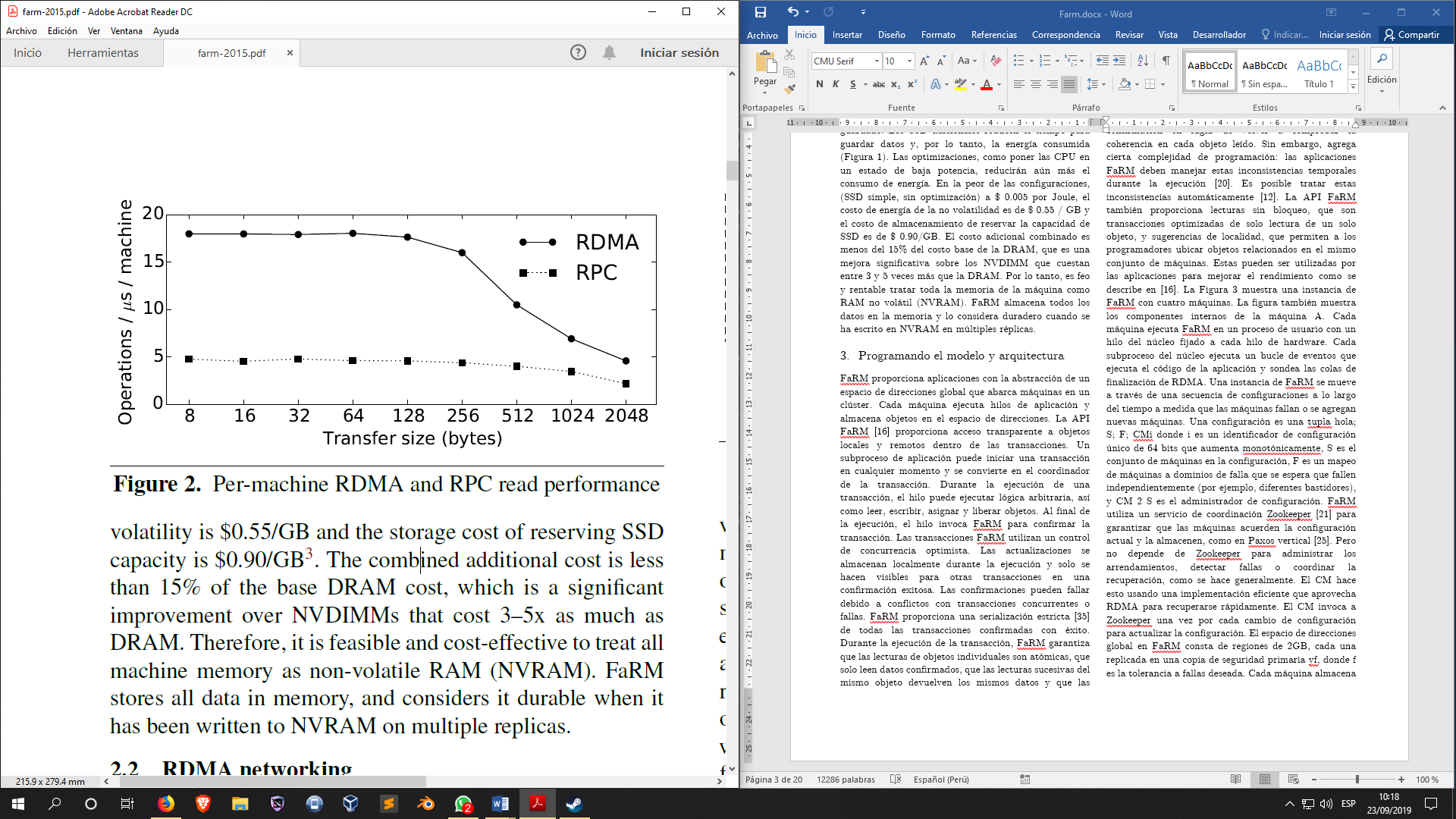
1. Tendencias del hardware

El diseño de FaRM está motivado por la disponibilidad de abundante, DRAM barata en máquinas de centros de datos. Un centro de datos típico tiene una configuración de 128–512 GB de DRAM por máquina de 2 sockets [29], y cada DRAM cuesta menos de $ 12/GB. Esto significa que un petabyte de DRAM requiere solo 2000 máquinas, y esto es suficiente para mantener los conjuntos de datos de muchas aplicaciones interesantes.

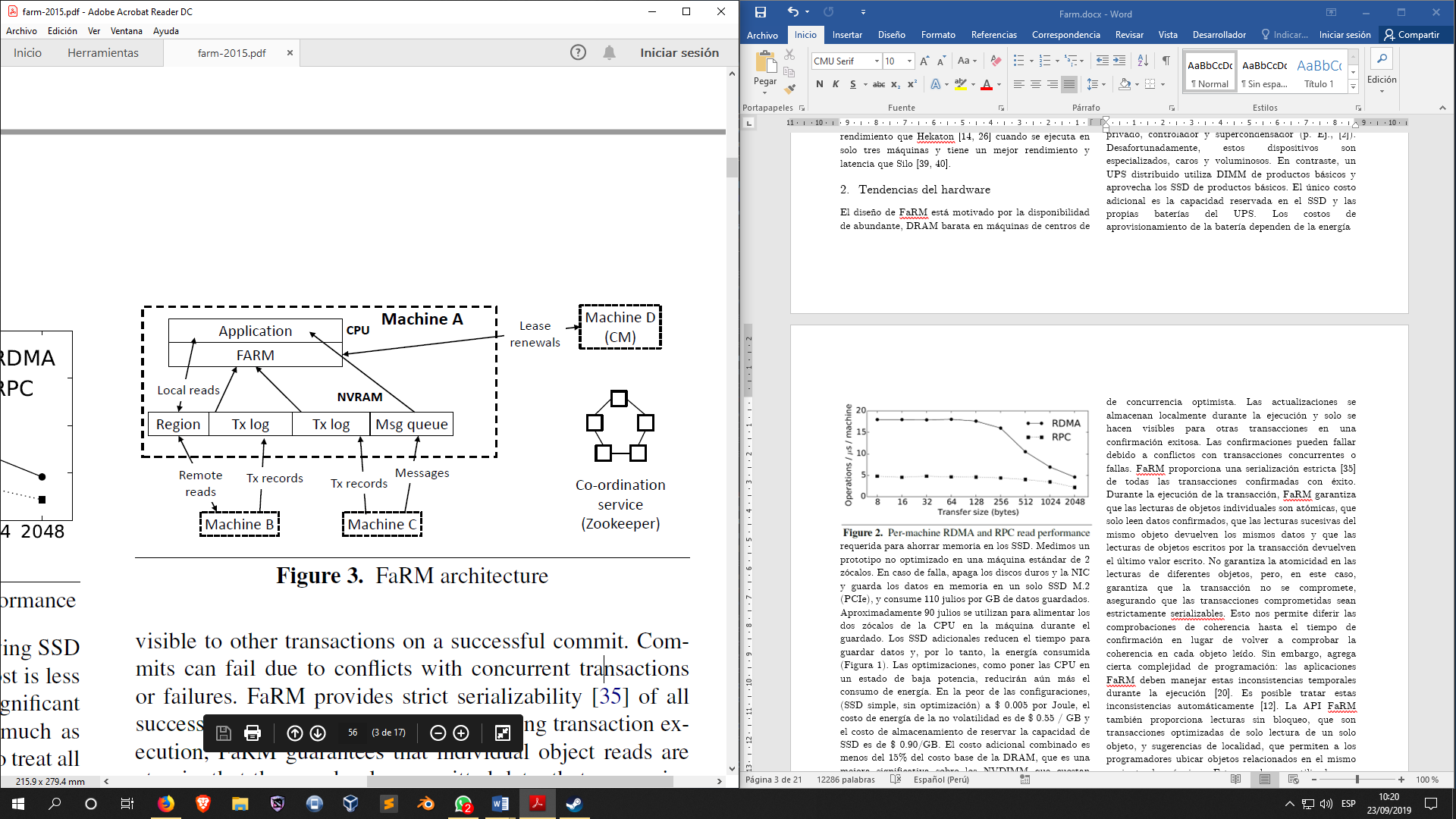


* 1. DRAM no volatil

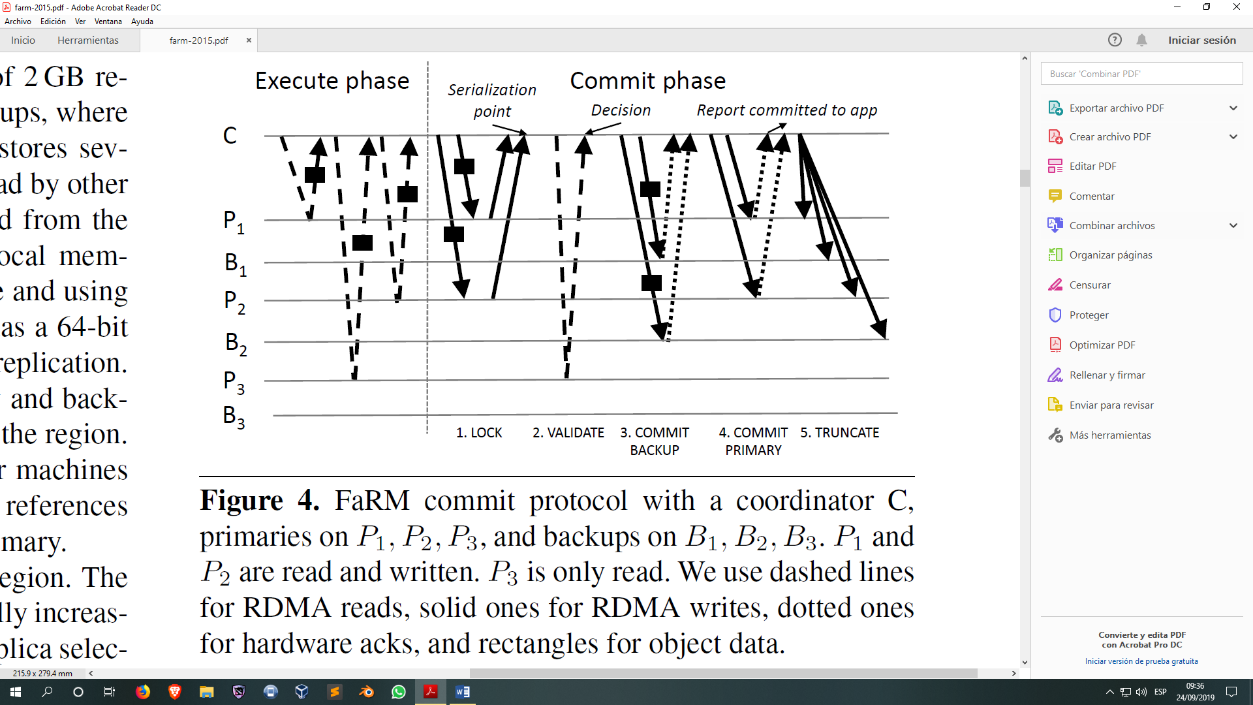
Una “fuente de alimentación ininterrumpida distribuida (UPS)” aprovecha la amplia disponibilidad de baterías de iones de litio para reducir el costo de un UPS de centro de datos sobre un enfoque centralizado tradicional que utiliza baterías de plomo-ácido. Por ejemplo, la especificación Open CloudServer (OCS) de Microsoft incluye Local Energy Storage (LES) [30, 36], que integra baterías de Liion con las unidades de suministro de energía en cada chasis de 24 máquinas dentro de un bastidor. El costo estimado de LES UPS es menos de $ 0.005 por Joule. Este enfoque es más confiable que un UPS tradicional: las baterías de iones de litio están sobreaprovisionadas con múltiples celdas independientes, y cualquier falla de la batería afecta solo una parte de un bastidor. Un UPS distribuido hace que la DRAM sea duradera. Cuando ocurre una falla de energía, el UPS distribuido guarda el contenido de la memoria en un SSD básico utilizando la energía de la batería. Esto no solo mejora el rendimiento de casos comunes al evitar escrituras síncronas en SSD, sino que también preserva la vida útil de la SSD al escribir solo cuando ocurren fallas. Un enfoque alternativo es utilizar DIMM no volátiles (NVDIMM), que contienen su propio flash privado, controlador y supercondensador (p. Ej., [2]). Desafortunadamente, estos dispositivos son especializados, caros y voluminosos. En contraste, un UPS distribuido utiliza DIMM de productos básicos y aprovecha los SSD de productos básicos. El único costo adicional es la capacidad reservada en el SSD y las propias baterías del UPS. Los costos de aprovisionamiento de la batería dependen de la energía

requerida para ahorrar memoria en los SSD. Medimos un prototipo no optimizado en una máquina estándar de 2 zócalos. En caso de falla, apaga los discos duros y la NIC y guarda los datos en memoria en un solo SSD M.2 (PCIe), y consume 110 julios por GB de datos guardados. Aproximadamente 90 julios se utilizan para alimentar los dos zócalos de la CPU en la máquina durante el guardado. Los SSD adicionales reducen el tiempo para guardar datos y, por lo tanto, la energía consumida (Figura 1). Las optimizaciones, como poner las CPU en un estado de baja potencia, reducirán aún más el consumo de energía. En la peor de las configuraciones, (SSD simple, sin optimización) a $ 0.005 por Joule, el costo de energía de la no volatilidad es de $ 0.55 / GB y el costo de almacenamiento de reservar la capacidad de SSD es de $ 0.90/GB. El costo adicional combinado es menos del 15% del costo base de la DRAM, que es una mejora significativa sobre los NVDIMM que cuestan entre 3 y 5 veces más que la DRAM. Por lo tanto, es feo y rentable tratar toda la memoria de la máquina como RAM no volátil (NVRAM). FaRM almacena todos los datos en la memoria y lo considera duradero cuando se ha escrito en NVRAM en múltiples réplicas.

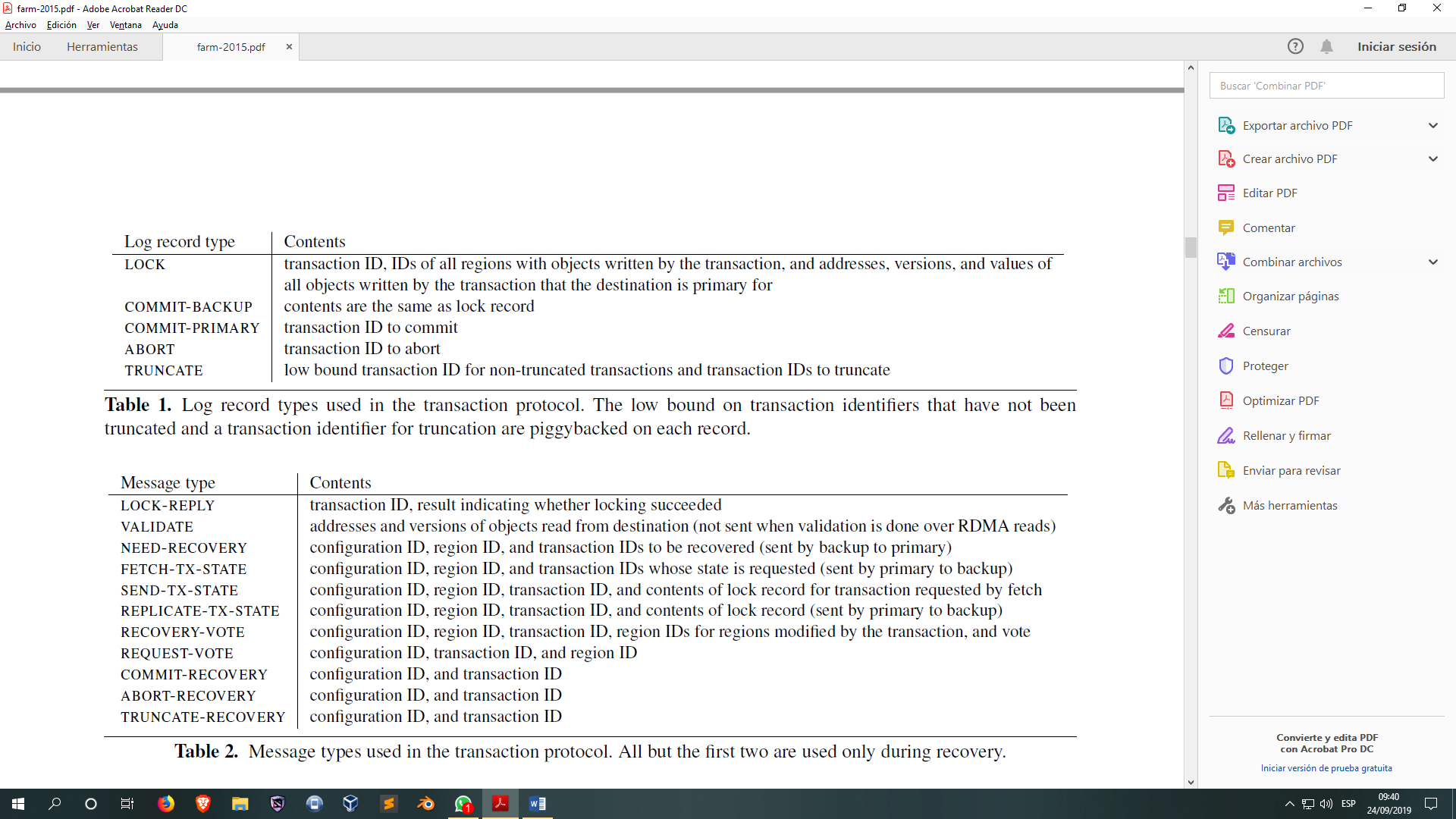
1. Programando el modelo y arquitectura

FaRM proporciona aplicaciones con la abstracción de un espacio de direcciones global que abarca máquinas en un clúster. Cada máquina ejecuta hilos de aplicación y almacena objetos en el espacio de direcciones. La API FaRM [16] proporciona acceso transparente a objetos locales y remotos dentro de las transacciones. Un subproceso de aplicación puede iniciar una transacción en cualquier momento y se convierte en el coordinador de la transacción. Durante la ejecución de una transacción, el hilo puede ejecutar lógica arbitraria, así como leer, escribir, asignar y liberar objetos. Al final de la ejecución, el hilo invoca FaRM para confirmar la transacción. Las transacciones FaRM utilizan un control

de concurrencia optimista. **Las actualizaciones se almacenan localmente durante la ejecución y solo se hacen visibles para otras transacciones en una confirmación exitosa.** Las confirmaciones pueden fallar debido a conflictos con transacciones concurrentes o fallas. FaRM proporciona una serialización estricta [35] de todas las transacciones confirmadas con éxito. Durante la ejecución de la transacción, **FaRM garantiza que las lecturas de objetos individuales son atómicas, que solo leen datos confirmados, que las lecturas sucesivas del mismo objeto devuelven los mismos datos y que las lecturas de objetos escritos por la transacción devuelven el último valor escrito.** No garantiza la atomicidad en las lecturas de diferentes objetos, pero, en este caso, garantiza que **la transacción no se compromete,** asegurando que las transacciones comprometidas sean estrictamente serializables. Esto nos permite diferir las comprobaciones de coherencia hasta el tiempo de confirmación en lugar de volver a comprobar la coherencia en cada objeto leído. Sin embargo, agrega cierta complejidad de programación: las aplicaciones FaRM deben manejar estas inconsistencias temporales durante la ejecución [20]. Es posible tratar estas inconsistencias automáticamente [12]. La API FaRM también proporciona lecturas sin bloqueo, que son transacciones optimizadas de solo lectura de un solo objeto, y sugerencias de localidad, que permiten a los programadores ubicar objetos relacionados en el mismo conjunto de máquinas. Estas pueden ser utilizadas por las aplicaciones para mejorar el rendimiento como se describe en [16]. La Figura 3 muestra una instancia de FaRM con cuatro máquinas. La figura también muestra los componentes internos de la máquina A. Cada máquina ejecuta FaRM en un proceso de usuario con un hilo del núcleo fijado a cada hilo de hardware. Cada subproceso del núcleo ejecuta un bucle de eventos que

ejecuta el código de la aplicación y sondea las colas de finalización de RDMA. Una instancia de FaRM se mueve a través de una secuencia de configuraciones a lo largo del tiempo a medida que las máquinas fallan o se agregan nuevas máquinas. Una configuración es una tupla {i, S, F, CM} donde i es un identificador de configuración único de 64 bits que aumenta monotónicamente, S es el conjunto de máquinas en la configuración, F es un mapeo de máquinas a dominios de falla que se espera que fallen independientemente (por ejemplo, diferentes estantes), y CM ∈ S es el administrador de configuración. FaRM utiliza un servicio de coordinación Zookeeper [21] para garantizar que las máquinas acuerden la configuración actual y la almacenen, como en Paxos vertical [25]. Pero no depende de Zookeeper para administrar los arrendamientos, detectar fallas o coordinar la recuperación, como se hace generalmente. El CM hace esto usando una implementación eficiente que aprovecha RDMA para recuperarse rápidamente. El CM invoca a Zookeeper una vez por cada cambio de configuración para actualizar la configuración. El espacio de direcciones global en FaRM consta de regiones de 2GB, cada una replicada en una copia de seguridad primaria y f respaldos, donde f es la tolerancia a fallas deseada. Cada máquina almacena varias regiones en DRAM no volátil que pueden leer otras máquinas utilizando RDMA. Los objetos siempre se leen desde la copia primaria de la región que lo contiene, utilizando accesos a la memoria local si la región está en la máquina local y utilizando lecturas RDMA unilaterales si son remotas. Cada objeto tiene una versión de 64 bits que se usa para el control y la replicación de concurrencia. El CM mantiene la asignación de un identificador de región a su primario y copias de seguridad y se replica con la región. Estas asignaciones se obtienen a pedido de otras máquinas y se almacenan en caché mediante subprocesos junto con las referencias RDMA necesarias para emitir lecturas RDMA unilaterales en el primario. Las máquinas contactan al CM para asignar una nueva región. El CM asigna un identificador de región de un contador monotónicamente creciente y selecciona réplicas para la región. La selección de réplicas equilibra el número de regiones almacenadas en cada máquina sujeta a las restricciones de que hay suficiente capacidad, cada réplica se encuentra en un dominio de falla diferente y la región se coloca con una región de destino cuando la aplicación específica una restricción de localidad. Luego envía un mensaje de preparación a las réplicas seleccionadas con el identificador de región. Si todas las réplicas informan éxito en la asignación de la región, el CM envía un mensaje de confirmación a todas ellas. Este protocolo de dos fases asegura que un mapeo sea válido y se replique en todas las réplicas de la región antes de que se use. Este enfoque centralizado proporciona más flexibilidad para satisfacer las limitaciones de independencia y localidad de fallas que nuestro enfoque anterior basado en hashing consistente [16]. También hace que sea más fácil equilibrar la carga entre máquinas y operar cerca de su capacidad. Con regiones de 2GB, esperamos hasta 250 regiones en una máquina típica y, por lo tanto, un solo CM podría manejar la asignación de regiones para miles de máquinas. Cada máquina también almacena buffers de anillo que implementan colas FIFO [16]. Se usan como registros de transacciones o como colas de mensajes. Cada par emisor-receptor tiene su propio registro y cola de mensajes, que se encuentran físicamente en el receptor. El remitente agrega registros al registro utilizando las escrituras RDMA unilaterales en su cola. Estas escrituras son reconocidas por la NIC sin involucrar a la CPU del receptor. El receptor sondea periódicamente el encabezado del registro para procesar los registros. Actualiza perezosamente al remitente cuando trunca el registro, permitiendo que el remitente reutilice el espacio en el búfer de anillo.

1. Transacciones distribuidas y su replicación

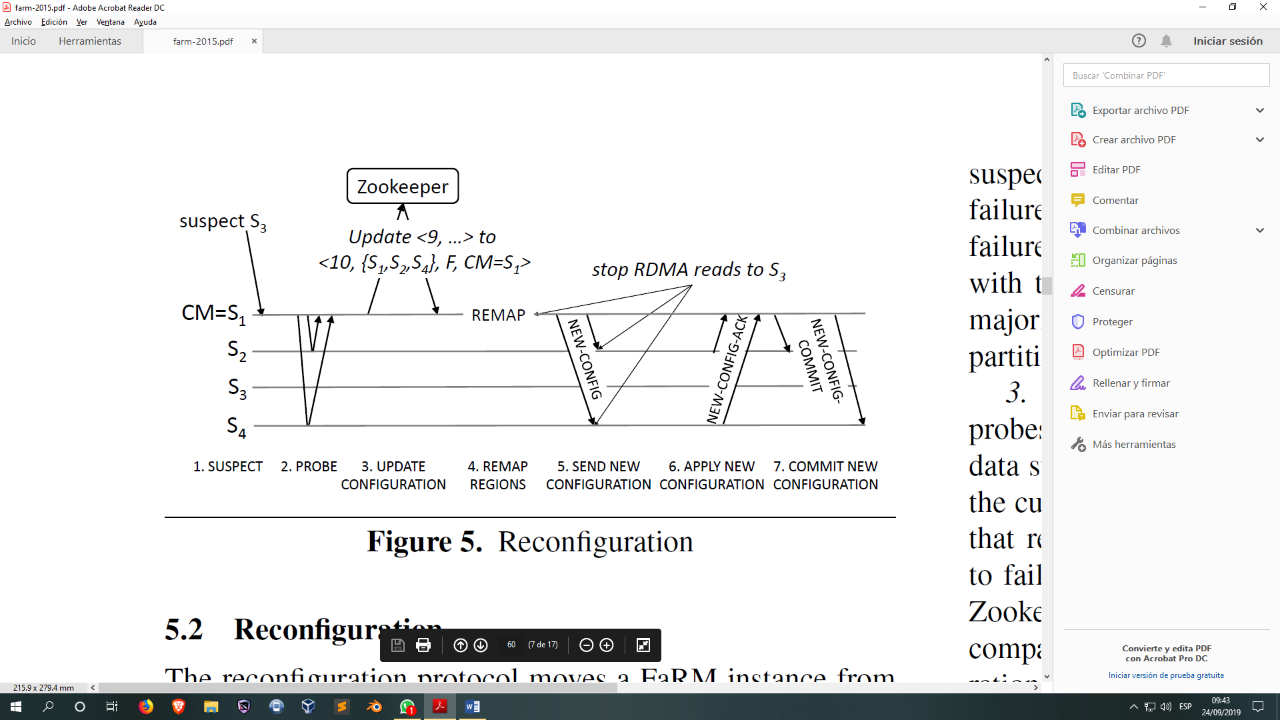
FaRM integra los protocolos de transacción y replicación para mejorar el rendimiento. Utiliza menos mensajes que los protocolos tradicionales y aprovecha las lecturas y escrituras RDMA unilaterales para la eficiencia de la CPU y la baja latencia. FaRM utiliza la replicación de copia de seguridad primaria en DRAM no volátil tanto para registros de datos como de transacciones, y utiliza coordinadores de transacciones no replicados que se comunican directamente con primarios y copias de seguridad. Utiliza un control de concurrencia optimista con validación de lectura, como en algunos sistemas de

memoria transaccional de software (por ejemplo, TL2 [15]). La Figura 4 muestra la línea de tiempo para una transacción FaRM y las tablas 1 y 2 enumeran todos los tipos de registros y mensajes de registro utilizados en el protocolo de transacción. Durante la fase de ejecución, las transacciones usan RDMA unilateral para leer objetos y almacenan localmente las escrituras. El coordinador también registra las direcciones y versiones de todos los objetos accedidos. Para las primarias y las copias de seguridad en la misma máquina que el coordinador, las lecturas y escrituras de objetos en el registro utilizan accesos de memoria local en lugar de RDMA. Al final de la ejecución, FaRM intenta confirmar la transacción ejecutando los siguientes pasos:

1. Bloqueo. El coordinador escribe un registro de BLOQUEO en el registro de cada máquina que es primaria para cualquier objeto escrito. Contiene las versiones y los nuevos valores de todos los objetos escritos en ese primario, así como la lista de todas las regiones con objetos escritos. Los primarios procesan estos registros al intentar bloquear los objetos en las versiones especificadas utilizando compareand-swap, y envían un mensaje informando si todos los bloqueos se tomaron con éxito. El bloqueo puede fallar si alguna versión del objeto cambió desde que fue leída por la transacción, o si el objeto está actualmente bloqueado por otra transacción. En este caso, el coordinador cancela la transacción. Escribe un registro de aborto en todas las primarias y devuelve un error a la aplicación.

2. Validar. El coordinador realiza la validación de lectura leyendo, desde sus primarias, las versiones de todos los objetos que fueron leídos pero no escritos por la transacción. Si algún objeto ha cambiado, la validación falla y la transacción se cancela. La validación usa lecturas RDMA unilaterales por defecto. Para las primarias que contienen más de tr objetos, la validación se realiza a través de RPC. El umbral tr (actualmente 4) refleja el costo de CPU de un RPC en relación con una lectura RDMA.

3. Confirmar copias de seguridad. El coordinador escribe un registro de COMMITBACKUP en los registros no volátiles en cada copia de seguridad y luego espera un ataque del hardware de la NIC sin interrumpir la CPU de la copia de seguridad. El registro de registro COMMIT-BACKUP tiene la misma carga útil que un registro LOCK.

4. Cometer primarias. Después de que todas las escrituras de COMPROMISO-COPIA DE SEGURIDAD hayan sido atacadas, el coordinador escribe un registro COMUNITARIO en los registros de cada primaria. Informa la finalización de la aplicación al recibir al menos un reconocimiento de hardware para dicho registro, o si escribió uno localmente. Las primarias procesan estos registros actualizando los objetos en su lugar, incrementando sus versiones y desbl

oqueándolos, lo que expone las escrituras confirmadas por la transacción.

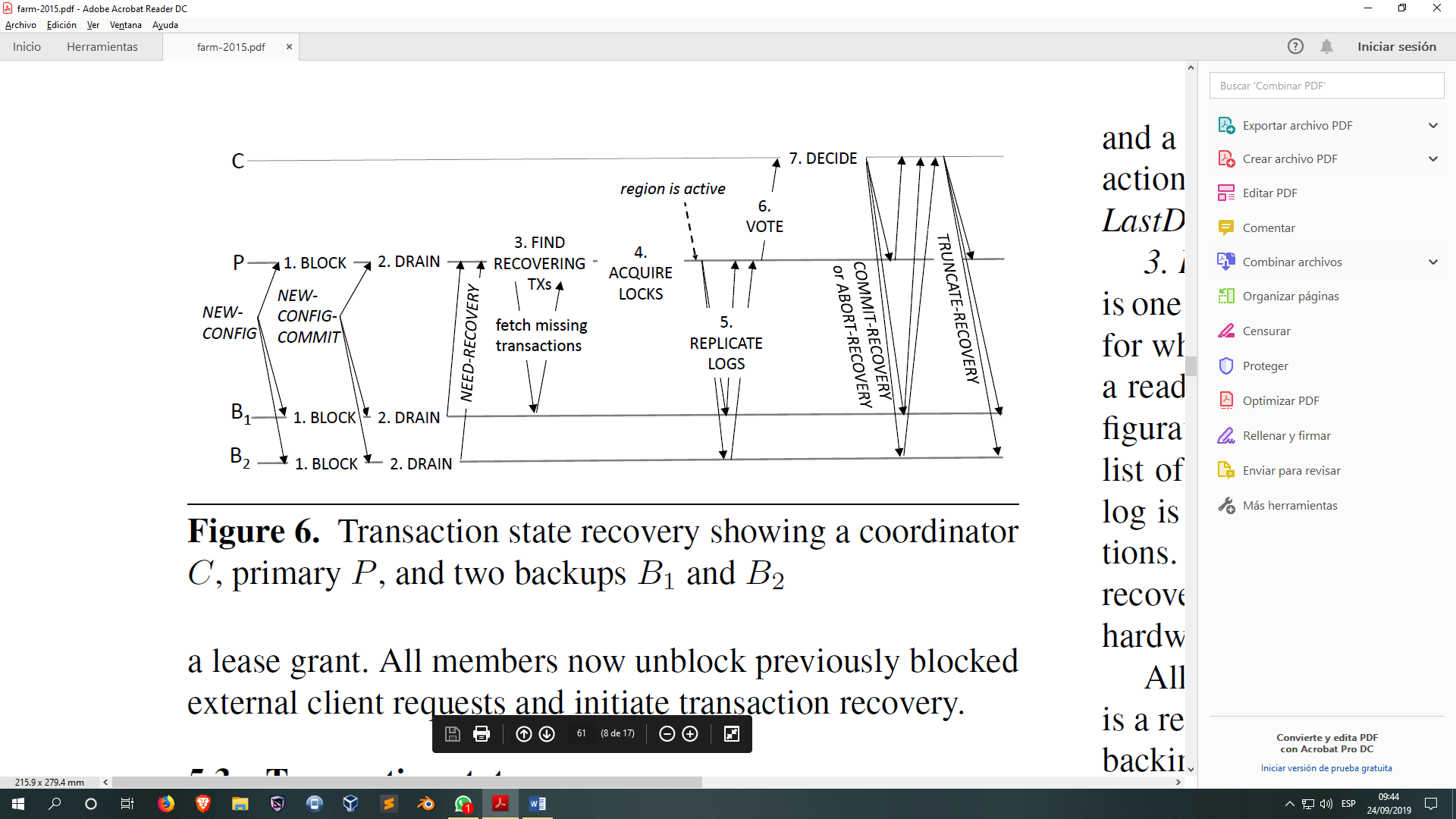
5. Truncar. Las copias de seguridad y las primarias mantienen los registros en sus registros hasta que se truncan. El coordinador trunca los registros en las primarias y las copias de seguridad perezosamente después de recibir los reconocimientos de todas las primarias. Para ello, lleva a cuestas identificadores de transacciones truncadas en otros registros. Las copias de seguridad aplican las actualizaciones a sus copias de los objetos en el momento del truncamiento.

1. Recuperación de fallas

FaRM proporciona durabilidad y alta disponibilidad mediante la replicación. Suponemos que las máquinas pueden fallar al estrellarse, pero pueden recuperarse sin perder el contenido de DRAM no volátil. Confiamos en la deriva del reloj delimitada por seguridad y eventualmente mensaje retrasado retrasos para la vida. Ofrecemos durabilidad para todas las transacciones comprometidas, incluso si todo el clúster falla o pierde energía: todo el estado comprometido se puede recuperar de regiones y registros almacenados en no volátiles DRACMA. Aseguramos durabilidad incluso si a lo sumo f réplicas por objeto pierden el contenido de DRAM no volátil. FaRM puede también mantener disponibilidad con fallas y particiones de red siempre que exista una partición que contenga una mayoría de las máquinas que permanecen conectadas entre sí y a una La mayoría de las réplicas en el servicio Zookeeper y la partición contiene al menos una réplica de cada objeto. La recuperación de fallas en FaRM tiene cinco fases que se describen a continuación: detección de fallas, reconfiguración, recuperación del estado de la transacción, recuperación de datos a granel y recuperación del estado del asignador.

* 1. Detección de fallas

FaRM utiliza arrendamientos [18] para detectar fallas. Cada máquina (que no sea el CM) tiene un contrato de arrendamiento en el CM y el CM tiene un contrato de arrendamiento en cualquier otra máquina. El vencimiento de cualquier arrendamiento desencadena la recuperación de fallas. Los arrendamientos se otorgan mediante un apretón de manos de 3 vías. Cada máquina envía una solicitud de arrendamiento al CM y responde con un mensaje que actúa como una concesión de arrendamiento a la máquina y una solicitud de arrendamiento del CM. Luego, la máquina responde con una concesión de arrendamiento al CM. Los arrendamientos FaRM son extremadamente cortos, lo cual es clave para una alta disponibilidad. Bajo una carga pesada, FaRM puede usar arrendamientos de 5 ms para un clúster de 90 máquinas sin falsos positivos. Los grupos significativamente más grandes pueden requerir una jerarquía de dos niveles, que en el peor de los casos duplicaría el tiempo de detección de fallas. Lograr arrendamientos cortos bajo carga requirió una implementación cuidadosa. FaRM utiliza pares de colas dedicados para arrendamientos para evitar retrasar los mensajes de arrendamiento en una cola compartida detrás de otros tipos de mensajes. El uso de un transporte confiable requeriría un par de colas adicional en el CM para cada máquina. Esto daría lugar a un bajo rendimiento debido a fallas de capacidad en la caché de pares de cola de la NIC [16]. En cambio, el administrador de arrendamiento utiliza verbos de envío y recepción Infiniband con el transporte de datagramas no confiable sin conexión, que requiere espacio para solo un par de colas adicional en la NIC. Por defecto, la renovación del contrato de arrendamiento se intenta cada 1 = 5 del período de vencimiento del contrato de arrendamiento para dar cuenta de la posible pérdida de mensajes. La renovación del arrendamiento también debe programarse en la CPU de manera oportuna. FaRM utiliza un subproceso de administrador de arrendamiento dedicado que se ejecuta con la máxima prioridad de espacio de usuario (31 en Windows). El subproceso de administrador de arrendamiento no está anclado a ningún subproceso de hardware y utiliza interrupciones en lugar de sondeo para evitar privar a las tareas críticas

del sistema operativo que deben ejecutarse periódicamente en cada subproceso de hardware. Esto aumenta la latencia del mensaje en unos pocos microsegundos, lo que no es problemático para los arrendamientos. Además, no asignamos subprocesos FaRM a dos subprocesos de hardware en cada máquina, dejándolos para el administrador de arrendamiento. Nuestras mediciones muestran que el administrador de arrendamiento generalmente se ejecuta en estos subprocesos de hardware sin afectar a otros subprocesos de FaRM, pero a veces está precedido por tareas de mayor prioridad que hacen que se ejecute en otros subprocesos de hardware. Por lo tanto, fijar el administrador de arrendamientos a un hilo de hardware probablemente resulte en falsos positivos cuando se usan arrendamientos cortos. Finalmente, asignamos previamente toda la memoria utilizada por el administrador de arrendamiento durante la inicialización y buscamos y fijamos todo el código que usa para evitar retrasos debido a la administración de la memoria.

* 1. Reconfiguración

El protocolo de reconfiguración mueve una instancia de FaRM de una configuración a la siguiente. El uso de operaciones RDMA unilaterales es importante para lograr un buen rendimiento, pero impone nuevos requisitos en el protocolo de reconfiguración. Por ejemplo, una técnica común para lograr consistencia es usar arrendamientos [18]: los servidores verifican si tienen un arriendo para un objeto antes de responder a las solicitudes de acceso al objeto. Si un servidor es desalojado de la configuración, el sistema garantiza que los objetos que almacena no pueden ser mutados hasta después de que expire su arrendamiento (por ejemplo, [7]). FaRM utiliza esta técnica cuando atiende solicitudes de clientes externos que se comunican con el sistema mediante mensajes. Pero dado que las máquinas en la configuración FaRM leen objetos utilizando lecturas RDMA sin involucrar a la CPU remota, la CPU del servidor no puede verificar si tiene el contrato de arrendamiento. El hardware actual de la NIC no admite arrendamientos y no está claro si lo hará en el futuro. Resolvemos este problema implementando una membresía precisa [10]. Después de una falla, todas las máquinas en una nueva configuración deben acordar su membresía antes de permitir mutaciones de objetos. Esto permite que FaRM realice la verificación en el cliente en lugar de hacerlo en el servidor. Las máquinas en la configuración no emiten solicitudes RDMA a las máquinas que no están en ella, y las respuestas a las lecturas RDMA y acks para las escrituras RDMA desde máquinas que ya no están en la configuración se ignoran. La Figura 5 muestra un ejemplo de línea de tiempo de reconfiguración que consta de los siguientes pasos:

1. Sospechoso. Cuando un contrato de arrendamiento de una máquina caduca en el CM, sospecha que esa máquina está fallando e inicia la reconfiguración. En este punto, comienza a bloquear todas las solicitudes de clientes externos. Si una máquina que no es CM sospecha que el CM falla debido a una expiración del arrendamiento, primero le pide a uno de un pequeño número de “CM de respaldo” que inicie la reconfiguración (los k sucesores del CM que usan hashing constante). Si la configuración no cambia después de un período de tiempo de espera, entonces intenta la reconfiguración misma. Este diseño evita una gran cantidad de intentos simultáneos de reconfiguración si el CM falla. En todos los casos, la máquina que inicia la reconfiguración intentará convertirse en el nuevo CM como parte de la reconfiguración.

2. Sonda. El nuevo CM emite una lectura RDMA a todas las máquinas en la configuración, excepto la máquina que se sospecha. También se sospecha de cualquier máquina para la cual falla la lectura. Estas sondas de lectura permiten el manejo de fallas correlacionadas que afectan a varias máquinas, por ejemplo, fallas de energía y conmutadores, por una sola reconfiguración. El nuevo CM continúa con la reconfiguración solo si obtiene respuestas para la mayoría de las sondas. Esto garantiza que si la red está particionada, el CM no estará en la partición más pequeña.

3. Actualización de la configuración. Después de recibir respuestas a las sondas, el nuevo CM intenta actualizar los datos de configuración almacenados en Zookeeper a {c + 1; S; F; }, donde c es el identificador de configuración actual, S es el conjunto de máquinas que respondieron a las sondas, F es la asignación de máquinas a dominios de falla y es su propio identificador. Utilizamos los números de secuencia de Zookeeper znode para implementar una comparación atómica y un intercambio que tiene éxito solo si la configuración actual sigue siendo c. Esto garantiza que solo una máquina pueda mover con éxito el sistema a la configuración con el identificador c + 1 (y convertirse en CM) incluso si varias máquinas intentan simultáneamente un cambio de configuración desde la configuración con el identificador c.

4. Remapear regiones. El nuevo CM luego reasigna regiones previamente asignadas a máquinas fallidas para restaurar el número de réplicas a f + 1. Intenta equilibrar la carga y satisfacer las sugerencias de localidad específicas de la aplicación sujetas a restricciones de independencia de capacidad y falla. Para las primarias fallidas, siempre promueve una copia de seguridad sobreviviente como la nueva primaria para reducir el tiempo de recuperación. Si detecta regiones que perdieron todas sus réplicas o no hay espacio para volver a replicar regiones, indica un error.

5. Enviar nueva configuración. Después de reasignar regiones, el CM envía un mensaje NEW-CONFIG a todas las máquinas en la configuración con el identificador de configuración, su propio identificador, los identificadores de las otras máquinas en la configuración y todas las nuevas asignaciones de regiones a máquinas. NEW-CONFIG también restablece el protocolo de arrendamiento si el CM ha cambiado: actúa como una solicitud de arrendamiento del nuevo CM a cada máquina. Si el CM no cambia, el intercambio de arrendamiento continúa durante la reconfiguración para detectar fallas adicionales rápidamente.

6. Aplicar nueva configuración. Cuando una máquina recibe una NUEVA CONFIGURACIÓN con un identificador de configuración que es mayor que el suyo, actualiza su identificador de configuración actual y su copia en caché de las asignaciones de región, y asigna espacio para contener cualquier nueva réplica de región asignada. Desde este punto, no emite nuevas solicitudes a máquinas que no están en la configuración y rechaza las respuestas de lectura y escritura de esas máquinas. También comienza a bloquear solicitudes de clientes externos. Las máquinas responden al CM con un mensaje NEW-CONFIG-ACK. Si el CM ha cambiado, esto otorga un arrendamiento al CM y solicita un arrendamiento.

7. Confirmar nueva configuración. Una vez que el CM recibe mensajes NEW-CONFIG-ACK de todas las máquinas en la configuración, espera para asegurarse de que los arrendamientos otorgados en configuraciones anteriores a máquinas que ya no estén en la configuración hayan expirado. El CM luego envía un NEW-CONFIGCOMMIT a todos los miembros de la configuración que también actúa como concesión de arrendamiento. Todos los miembros ahora desbloquean solicitudes de clientes externos previamente bloqueadas e inician la recuperación de la transacción.

* 1. Recuperación del estado de la transacción

FaRM recupera el estado de la transacción después de un cambio de configuración utilizando los registros distribuidos entre las réplicas de los objetos modificados por una transacción. Esto implica recuperar el estado tanto en las réplicas de los objetos modificados por la transacción como en el coordinador para decidir el resultado de la transacción. La Figura 6 muestra un ejemplo de cronograma de recuperación de transacciones. FaRM logra una recuperación rápida mediante la distribución de trabajo entre subprocesos y máquinas en el clúster. El drenaje (paso 2) se realiza para todos los registros de mensajes en paralelo. Los pasos 1 y 3–5 se realizan para todas las regiones en paralelo. Los pasos 6–7 se realizan para todas las transacciones de recuperación en paralelo.

1. Bloquear el acceso a las regiones en recuperación. Cuando falla la primaria de una región, una de las copias de seguridad se promociona para que sea la nueva primaria durante la reconfiguración. No podemos permitir el acceso a la región hasta que todas las transacciones que la hayan actualizado se hayan reflejado en la nueva primaria. Hacemos esto bloqueando las solicitudes de punteros locales y referencias de RDMA a la región hasta el paso 4 cuando se han adquirido todos los bloqueos de escritura para todas las transacciones de recuperación que actualizaron la región.

2. Drene los troncos. Las escrituras RDMA unilaterales también afectan la recuperación de transacciones. Un enfoque general para la coherencia entre configuraciones es rechazar mensajes de configuraciones antiguas. FaRM no puede usar este enfoque porque las NIC reconocen los registros COMMIT-BACKUP y COMMIT-PRIMARY escritos en los registros de transacciones, independientemente de la configuración en la que se emitieron. Dado que los coordinadores solo esperan estos reconocimientos antes de exponer las actualizaciones y reportar el éxito a la aplicación, las máquinas no siempre pueden rechazar registros de configuraciones anteriores cuando las procesan. Resolvemos este problema drenando registros para garantizar que todos los registros relevantes se procesen durante la recuperación: todas las máquinas procesan todos los registros en sus registros cuando reciben un mensaje NEW-CONFIGCOMMIT. Registran el identificador de configuración en una variable LastDrained cuando terminan. Las transacciones FaRM tienen identificadores únicos hc; metro; t; li asignado al comienzo de la confirmación que codifica la configuración c en la que se inició la confirmación, el identificador de máquina m del coordinador, el identificador de subproceso t del coordinador y un identificador único local de subproceso l. Los registros de registro para transacciones con identificadores de configuración inferiores o iguales a LastDrained se rechazan.

3. Encuentra las transacciones de recuperación. Una transacción de recuperación es aquella cuya fase de confirmación abarca cambios de configuración y para la cual alguna réplica de un objeto escrito, alguna primaria de un objeto leído o el coordinador ha cambiado debido a la reconfiguración. Durante el drenaje del registro, se examina el identificador de transacción y la lista de identificadores de región actualizados en cada registro de registro en cada registro para determinar el conjunto de transacciones de recuperación. Solo las transacciones de recuperación pasan por la recuperación de transacciones en las primarias y las copias de seguridad, y los coordinadores rechazan los ataques de hardware solo para recuperar transacciones. Todas las máquinas deben acordar si una transacción dada es una transacción de recuperación o no. Logramos esto mediante la introducción de algunos metadatos adicionales en la comunicación durante la fase de reconfiguración. El CM lee la variable LastDrained en cada máquina como parte de la lectura de la sonda. Para cada región r cuya asignación ha cambiado desde LastDrained, el CM envía dos identificadores de configuración en el mensaje NEW-CONFIG a esa máquina. Estos son LastPrimaryChange [r], el último identificador de configuración cuando cambió el primario de r, y LastReplicaChange [r], el último identificador de configuración cuando cambió cualquier réplica de r. Una transacción que comenzó a confirmarse en la configuración c 􀀀 1 se está recuperando en la configuración c a menos que: para todas las regiones r que contengan objetos modificados por la transacción LastReplicaChange [r] <c, para todas las regiones r0 que contengan objetos leídos por la transacción LastPrimaryChange [r0] <c , y el coordinador no se ha eliminado de la configuración c. Los registros para una transacción de recuperación pueden distribuirse entre los registros de diferentes primarios y copias de seguridad actualizados por la transacción. Cada copia de seguridad de una región envía un mensaje NEEDRECOVERY al primario con el identificador de configuración, el identificador de región y los identificadores de las transacciones de recuperación que actualizaron la región.

4. Recuperación de bloqueo. El primario de cada región espera hasta que los registros de la máquina local se hayan agotado y se hayan recibido mensajes de NECESIDAD DE RECUPERACIÓN de cada copia de seguridad, para construir el conjunto completo de transacciones de recuperación que afectan a la región. Luego divide las transacciones por identificador en sus subprocesos de modo que cada subproceso t recupera el estado de las transacciones con el identificador de subprocesador coordinador t. Paralelamente, los subprocesos en el primario recuperan los registros de transacciones de las copias de seguridad que no están almacenadas localmente y luego bloquean los objetos modificados al recuperar las transacciones. Cuando se completa la recuperación de bloqueo para una región, la región está activa y los coordinadores locales y remotos pueden obtener punteros locales y referencias RDMA, lo que les permite leer objetos y enviar actualizaciones a esta región en paralelo con los pasos de recuperación posteriores.

5. Replicar registros de anotaciones. Los subprocesos en los registros de réplica primarios envían copias de seguridad del mensaje REPLICATE-TXSTATE para las transacciones que faltan. El mensaje contiene el identificador de región, el identificador de configuración actual y los mismos datos que el registro LOCK.

6. Vota. El coordinador de una transacción de recuperación decide si comprometer o cancelar la transacción en función de los votos de cada región actualizada por la transacción. Estos votos son enviados por las primarias de cada región. FaRM utiliza un hashing coherente para determinar el coordinador de una transacción, asegurando que todas las primarias acuerden independientemente la identidad del coordinador para una transacción de recuperación. El coordinador no cambia si la máquina en la que se ejecuta todavía está en la configuración, pero cuando un coordinador falla, la responsabilidad de coordinar sus transacciones de recuperación se extiende a través de las máquinas en el clúster. Los hilos en el primario envían mensajes de VOTO DE RECUPERACIÓN a sus hilos pares en el coordinador para cada transacción de recuperación que modificó la región. El voto es commit-primary si alguna réplica vio COMMIT-PRIMARY o COMMIT-RECOVERY. De lo contrario, vota commit-backup si alguna réplica vio COMMIT-BACKUP y no vio ABORTRECOVERY. De lo contrario, vota bloqueo si alguna réplica vio un registro de BLOQUEO y no ABORT-RECUPERACIÓN. De lo contrario, vota abortar. Los mensajes de votación incluyen el identificador de configuración, el identificador de región, el identificador de transacción y la lista de identificadores de región modificados por la transacción. Algunas primarias pueden no iniciar la votación de una transacción porque nunca recibieron un registro de la transacción o ya truncaron los registros de la transacción. El coordinador envía solicitudes de voto explícito a las primarias que aún no han votado dentro de un período de tiempo de espera (establecido en 250 \_s). El mensaje SOLICITAR VOTO incluye el identificador de configuración, el identificador de región y el identificador de transacción. Las primarias que tienen registros de registro para la transacción votan como antes después de esperar la replicación del registro para que se complete esa transacción. Las primarias que no tienen ningún registro de registro para la transacción se truncan si la transacción ya se ha truncado y se desconoce si no. Para determinar si una transacción ya se ha truncado, cada subproceso mantiene el conjunto de identificadores de transacciones cuyos registros se han truncado de sus registros. Este conjunto se mantiene compacto mediante el uso de un límite inferior en los identificadores de transacciones no truncadas. El límite inferior se actualiza en función de los límites inferiores de cada coordinador, que se complementan con los mensajes del coordinador y durante la reconfiguración.

7. Decidir. El coordinador decide comprometer una transacción si recibe un voto de confirmación primaria de cualquier región. De lo contrario, espera a que todas las regiones voten y se compromete si al menos una región votó commit-backup y todas las demás regiones modificadas por la transacción votaron bloqueo, commitbackup o truncado. De lo contrario, decide abortar. Luego envía COMMIT-RECOVERY o ABORT-RECOVERY a todas las réplicas participantes. Ambos mensajes incluyen el identificador de configuración y el identificador de transacción. COMMIT-RECOVERY se procesa de manera similar a COMMIT-PRIMARY si se recibe en una primaria y a COMMIT-BACKUP si se recibe en una copia de seguridad. ABORT-RECOVERY se procesa de manera similar a ABORT. Después de que el coordinador recibe los acuses de recibo de todas las primarias y copias de seguridad, envía un mensaje TRUNCATE-RECUPERACIÓN.

Exactitud. A continuación, damos una idea de cómo los diferentes pasos de la recuperación de transacciones aseguran una serialización estricta. La idea clave es que la recuperación conserva el resultado de las transacciones que se comprometieron o cancelaron anteriormente. Decimos que una transacción se confirma cuando una entidad primaria expone modificaciones de la transacción o el coordinador notifica a la aplicación que la transacción se confirmó. Una transacción se cancela cuando el coordinador envía un mensaje de cancelación o notifica a la aplicación que la transacción se ha cancelado. Para las transacciones cuyo resultado aún no se ha decidido, la recuperación puede comprometer o abortar la transacción, pero asegura que cualquier recuperación de fallas adicionales conserva el resultado. El resultado de las transacciones que no se están recuperando (paso 3) se decide utilizando el protocolo de caso normal (Sección 4). Entonces no los discutiremos más. Se garantiza que un registro de registro para una transacción de recuperación comprometida se procesará y aceptará antes o durante el drenaje del registro (paso 2). Esto es cierto porque las primarias exponen modificaciones solo después de procesar el registro COMMITPRIMARY. Si el coordinador notificó a la aplicación, debe haber recibido reconocimientos de hardware para todos los registros COMMITBACKUP y al menos para un registro COMMIT-PRIMARY antes de recibir NEW-CONFIG (porque ignora los reconocimientos después de cambiar la configuración). Por lo tanto, dado que la nueva configuración incluye al menos una réplica para cada región, al menos una réplica para al menos una región procesará los registros COMMIT-PRIMARY o COMMIT-BACKUP, y al menos una réplica para cada región procesará COMMITPRIMARY, COMMIT- COPIA DE SEGURIDAD o BLOQUEO de registros. Los pasos 3 y 4 aseguran que las primarias de las regiones modificadas por la transacción vean estos registros (a menos que se hayan truncado). Replican estos registros en las copias de seguridad (paso 5) para garantizar que la votación produzca los mismos resultados incluso si hay fallas posteriores. Luego, las primarias envían votos al coordinador en función de los registros que han visto (paso 6). El paso de decisión garantiza que el coordinador decide comprometer cualquier transacción que se haya comprometido previamente. Si alguna réplica trunca los registros de transacciones, todas las primarias votarán commit-primary, commit-backup o truncated. Al menos una primaria enviará un voto que no sea truncado porque de lo contrario la transacción no se recuperaría. Si ninguna réplica trunca los registros de transacciones, al menos una primaria votará commit-primary o commit-backup y las otras votarán commit-primary, commit-backup o lock. Del mismo modo, el coordinador decidirá abortar si la transacción se canceló previamente porque en este caso no habrá registros de confirmación primaria o de confirmación de confirmación o todas las réplicas habrán recibido ABORT-RECOVERY. El bloqueo del acceso a las regiones de recuperación (paso 1) y la recuperación de bloqueo (paso 4) garantizan que hasta que una transacción de recuperación se haya confirmado o cancelado, ninguna otra operación puede acceder a los objetos que modificó.

Actuación. FaRM utiliza varias optimizaciones para lograr una recuperación rápida de fallas. La identificación de las transacciones de recuperación limita el trabajo de recuperación solo a aquellas transacciones y regiones que fueron afectadas por la reconfiguración, lo que podría ser un pequeño subconjunto del total cuando falla una sola máquina en un clúster grande. Nuestros resultados indican que esto puede reducir el número de transacciones a recuperar en un orden de magnitud. El trabajo de recuperación en sí está paralelo a través de regiones, máquinas y subprocesos. Hacer que las regiones estén disponibles inmediatamente después de la recuperación del bloqueo mejora el rendimiento en primer plano ya que las nuevas transacciones que acceden a estas regiones no se bloquean por mucho tiempo. Específicamente, no necesitan esperar mientras se actualizan las nuevas réplicas de estas regiones, lo que requiere un movimiento masivo de datos a través de la red.

* 1. Recuperando datos

FaRM debe recuperar (volver a replicar) datos en nuevas copias de seguridad para una región para garantizar que pueda tolerar f fallas de réplica en el futuro. La recuperación de datos no es necesaria para reanudar el funcionamiento normal de los casos, por lo que la retrasamos hasta que todas las regiones se activen para minimizar el impacto en la recuperación de bloqueo de latencia crítica. Cada máquina envía un mensaje REGIONS-ACTIVE al CM cuando todas las regiones para las cuales es primaria se activan. Después de recibir todos los mensajes REGIONS-ACTIVE, el CM envía un mensaje ALL-REGIONS-ACTIVE a todas las máquinas en la configuración. En este punto, FaRM comienza la recuperación de datos para nuevas copias de seguridad en paralelo con las operaciones en primer plano. Una nueva copia de seguridad para una región inicialmente tiene una réplica de región local recién asignada y puesta a cero. Divide la región en hilos de trabajo que la recuperan en paralelo. Cada hilo emite operaciones RDMA unilaterales para leer un bloque a la vez desde el primario. Actualmente utilizamos bloques de 8 KB, que son lo suficientemente grandes como para usar la red de manera eficiente pero lo suficientemente pequeños como para no afectar el funcionamiento normal de la carcasa. Para reducir el impacto en el rendimiento en primer plano, la recuperación se planifica programando la próxima lectura para que comience en un punto aleatorio dentro de un intervalo después del inicio de la lectura anterior (establecida en 4 ms). Cada objeto recuperado debe ser examinado antes de ser copiado a la copia de seguridad. Si el objeto tiene una versión mayor que la versión local, la copia de seguridad bloquea la versión local con una comparación e intercambio, actualiza el estado del objeto y lo desbloquea. De lo contrario, el objeto ha sido o está siendo actualizado por una transacción que creó una versión mayor o igual a la recuperada, y el estado recuperado no se aplica.

* 1. Recuperando el estado del asignador

El asignador FaRM divide las regiones en bloques (1 MB) que se utilizan como losas para asignar objetos pequeños. Mantiene dos piezas de metadatos: encabezados de bloque, que contienen el tamaño del objeto, y listas libres de losas. Los encabezados de bloque se replican en copias de seguridad cuando se asigna un nuevo bloque. Esto garantiza que estén disponibles en el nuevo primario después de una falla. Como los encabezados de bloque se utilizan en la recuperación de datos, el nuevo primario los envía a todas las copias de seguridad inmediatamente después de recibir NEWCONFIG-COMMIT. Esto evita cualquier inconsistencia cuando el antiguo primario falla al replicar el encabezado del bloque. Las listas libres de losas se mantienen solo en la primaria para reducir los gastos generales de la asignación de objetos. Cada objeto tiene un bit en su encabezado que se establece mediante una asignación y se borra de forma gratuita durante la ejecución de la transacción. Este cambio en el estado del objeto se replica durante la confirmación de la transacción como se describe en la Sección 4. Después de una falla, las listas libres se recuperan en el nuevo primario escaneando los objetos en la región, que se encuentra en paralelo en todos los hilos de la máquina. Para minimizar el impacto en la recuperación del bloqueo de transacciones, la recuperación de la asignación comienza después de que se recibe ALL-REGIONS-ACTIVE y para minimizar el impacto en el trabajo en primer plano se escanea 100 objetos a la vez cada 100 μs. Las asignaciones de objetos se ponen en cola hasta que se recupera la lista libre de una losa.

1. Evaluación
   1. Preparación

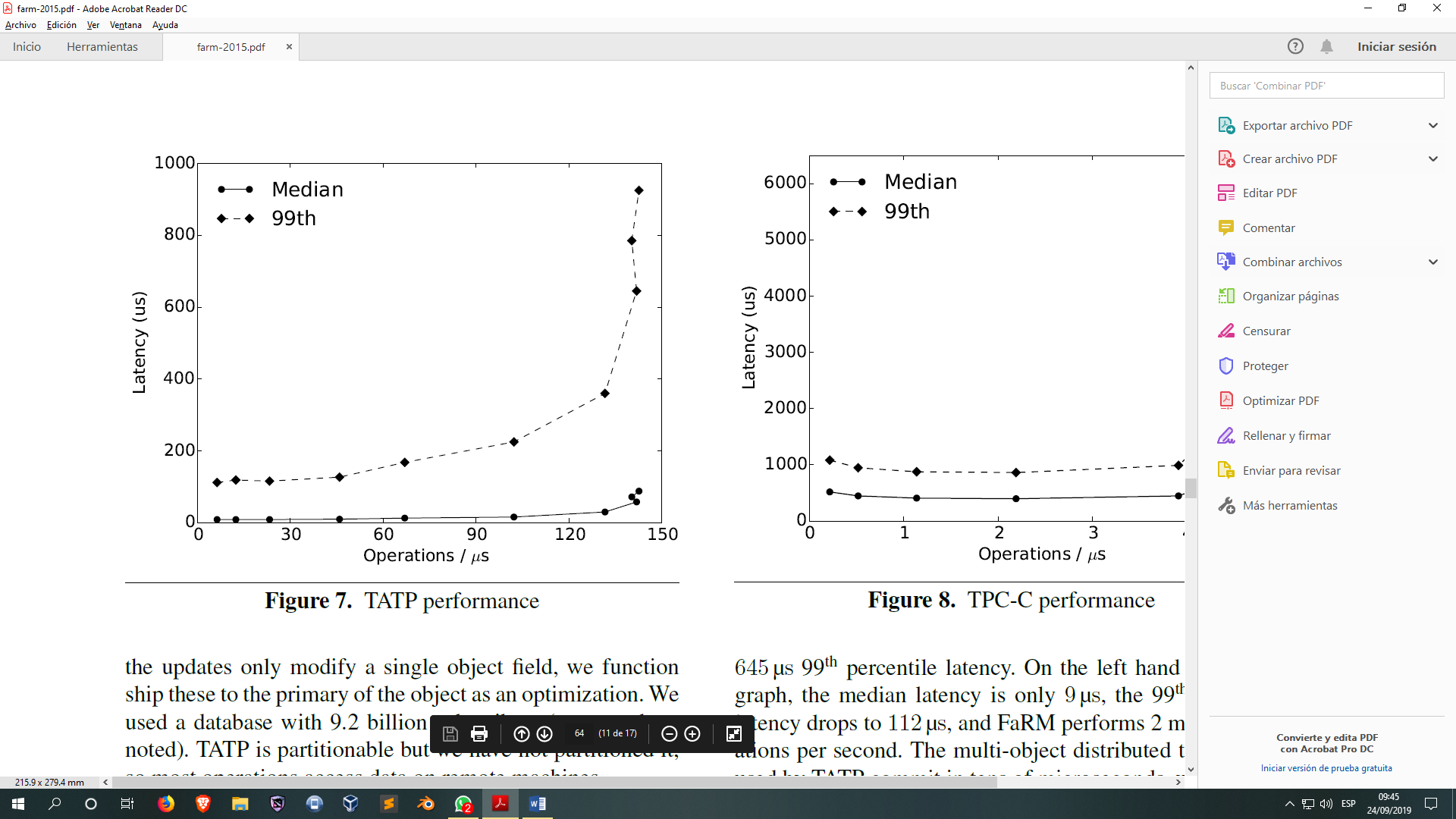
Nuestro banco de pruebas experimental consta de 90 máquinas utilizadas para un clúster FaRM y 5 máquinas para una instancia replicada de Zookeeper. Cada máquina tiene 256 GB de DRAM y dos CPU Intel E5-2650 de 8 núcleos con Windows Server 2012 R2. Permitimos el subprocesamiento múltiple y utilizamos los primeros 30 subprocesos para el trabajo en primer plano y los 2 subprocesos restantes para el administrador de arrendamiento. Las máquinas tienen dos NIC Infiniband Mellanox ConnectX-3 de 56 Gbps, cada una utilizada por subprocesos en un zócalo diferente, y están conectadas por un solo conmutador Mellanox SX6512 con ancho de banda de bisección completo. FaRM se configuró para usar la replicación de 3 vías (una primaria y dos copias de seguridad) con un tiempo de arrendamiento de 10 ms.

* 1. Benchmarks

Utilizamos dos puntos de referencia transaccionales para medir el rendimiento de FaRM. Implementamos ambos puntos de referencia en C ++ contra la API FaRM. Dado que FaRM usa un modelo simétrico para explotar la localidad, cada máquina ejecuta el código de referencia y almacena datos. Cada máquina ejecuta el código de referencia vinculado con el código de FaRM en el mismo proceso. En el futuro, compilaremos la aplicación desde un lenguaje seguro como SQL para evitar que los errores de la aplicación corrompan los datos. El procesamiento de transacciones de aplicaciones de telecomunicaciones (TATP) [32] es un punto de referencia para las bases de datos de memoria principal de alto rendimiento. Cada tabla de base de datos se implementa como una tabla hash FaRM [16]. TATP es de lectura dominada. El 70% de las operaciones son búsquedas de una sola fila que utilizan lecturas libres de bloqueo de FaRM [16]. Por lo general, se pueden realizar con una sola lectura RDMA y no requieren una fase de confirmación. El 10% de las operaciones leen 2–4 filas y requieren validación durante la fase de confirmación. El 20% restante de las operaciones son actualizaciones y requieren el protocolo de confirmación completo. Dado que el 70% de las actualizaciones solo modifican un campo de un solo objeto, las enviamos al primario del objeto como una optimización. Utilizamos una base de datos con 9.2 mil millones de suscriptores (excepto donde se indique). TATP es particionable pero no lo hemos particionado, por lo que la mayoría de las operaciones acceden a datos en máquinas remotas. TPC-C [38] es un punto de referencia de base de datos bien conocido con transacciones complejas que acceden a cientos de filas. Nuestra implementación utiliza un esquema con 16 índices. Doce de estos solo requieren consultas y actualizaciones desordenadas (puntuales) y se implementan como tablas hash FaRM. Cuatro de los índices también requieren consultas de rango. Estos se implementan utilizando el árbol B FaRM. El B-Tree almacena en caché los nodos internos en cada máquina y, por lo tanto, las búsquedas requieren una sola lectura FaRM RDMA en el caso común. Reservamos 8 GB por máquina para el caché. Utilizamos llaves de valla [17, 27] para garantizar la consistencia transversal, similar a Minuet [37]. Omitimos una descripción más detallada del árbol B por razones de espacio. Utilizamos una base de datos con 21,600 bodegas. Compartimos la mayoría de los índices de la tabla hash y los clientes por almacén, lo que significa que alrededor del 10% de todas las transacciones acceden a datos remotos. Según lo especificado por el punto de referencia, las transacciones de "nuevos pedidos" representan el 45% de la combinación de transacciones. Ejecutamos la combinación completa pero informamos el rendimiento como el número de "nuevos pedidos" comprometidos con éxito.

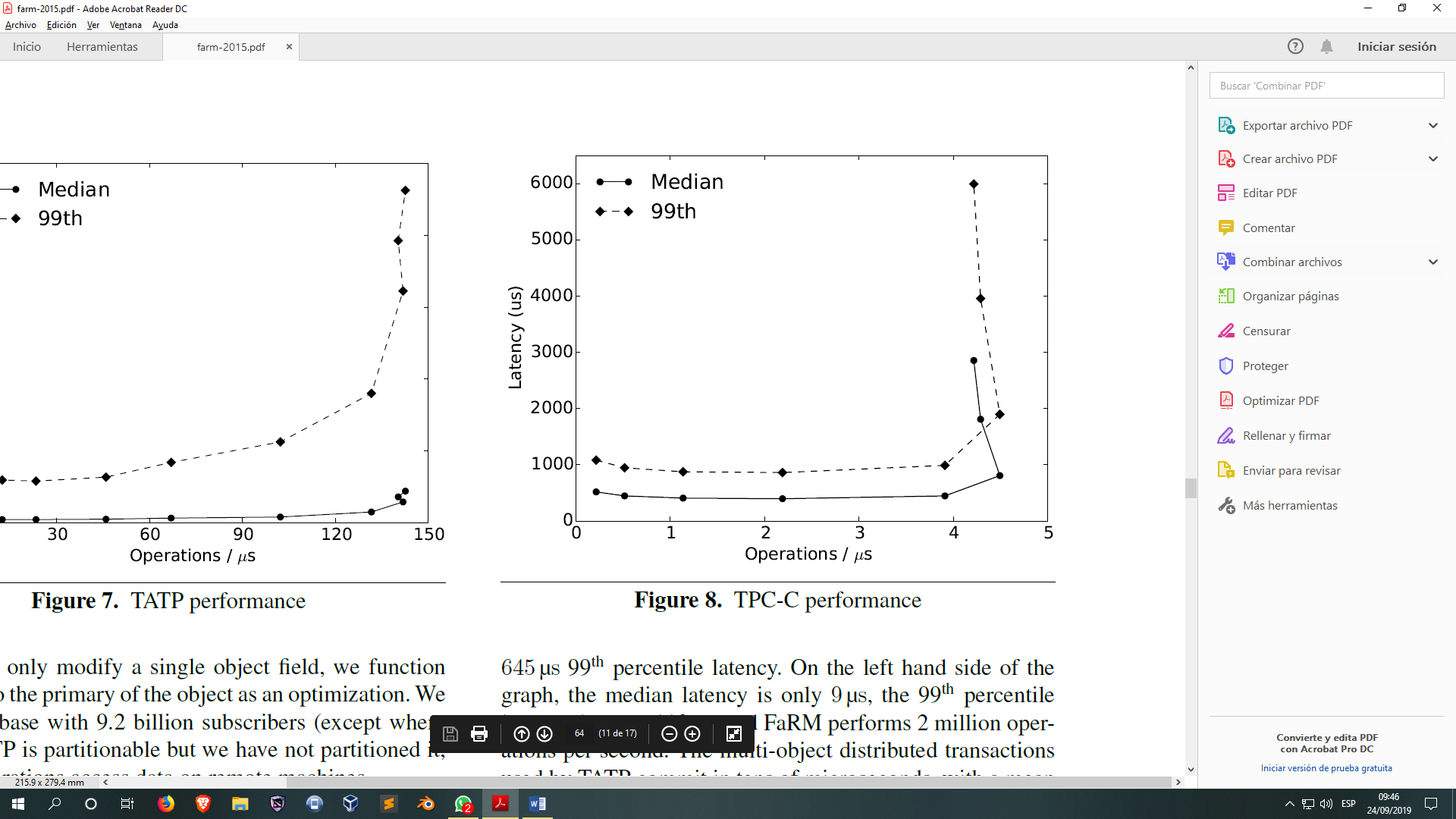
* 1. Rendimiento del caso normal

Presentamos el rendimiento del caso normal (sin fallas) de FaRM como curvas de rendimiento y latencia. Para cada punto de referencia, variamos la carga aumentando primero el número de subprocesos activos por máquina de 2 a 30 y luego aumentando la concurrencia por subproceso, hasta que se saturó el rendimiento. Tenga en cuenta que el extremo izquierdo de cada gráfico aún

muestra una concurrencia significativa y, por lo tanto, el rendimiento. No muestra la latencia mínima que puede lograr FaRM.

TATP. La Figura 7 muestra que FaRM realiza 140 millones de transacciones TATP por segundo con una latencia media de 58 μs y una latencia del percentil 99th de 645 μs. En el lado izquierdo del gráfico, la latencia media es de solo 9 μs, la latencia del percentil 99th cae a 112 μs y FaRM realiza 2 millones de operaciones por segundo. Las transacciones distribuidas de múltiples objetos utilizadas por TATP se comprometen en decenas de microsegundos, con una latencia de confirmación media de 19 μs en el rendimiento más bajo y 138 μs en el más alto. FaRM supera los resultados de TATP publicados para Hekaton [14, 26], un motor transaccional en memoria de una sola máquina, en un factor de 33. Los resultados de Hekaton se obtuvieron utilizando hardware diferente, pero esperamos un factor de mejora de 20 cuando se ejecuta Hekaton en uno de nuestras máquinas de banco de pruebas. En un experimento a menor escala, FaRM superó a Hekaton con solo tres máquinas. Además, FaRM admite conjuntos de datos mucho más grandes porque se amplía y proporciona una alta disponibilidad a diferencia de los sistemas de máquinas individuales.

TPC-C. Ejecutamos TPC-C durante 60 segundos e informamos latencia y rendimiento promedio durante ese período en la Figura 8. FaRM realiza hasta 4.5 millones de transacciones TPC-C de "nuevo pedido" por segundo con una latencia media de 808 μs y una latencia de percentil 99th de 1.9 ms. La latencia se puede reducir a la mitad con un pequeño impacto del 10% en el rendimiento. El mejor rendimiento de TPC-C publicado

que conocemos es de Silo [39, 40], que es un sistema en memoria de máquina única con registro en SSD FusionIO. El rendimiento de FaRM es 17 veces mayor que Silo sin registro, y su latencia en este nivel de rendimiento es 128 veces mejor que Silo con registro. Leer rendimiento. Aunque el objetivo de este documento es el rendimiento transaccional y la recuperación de fallos, también pudimos mejorar el rendimiento de solo lectura en relación con [16]. Ejecutamos una carga de trabajo de búsqueda de valor de clave con claves de 16 bytes y valores de 32 bytes y un patrón de acceso uniforme. Logramos un rendimiento de 790 millones de búsquedas / s con una latencia media de 23 μs y una latencia del percentil 99th de 73 μs. Esto mejora el rendimiento por máquina reportado previamente para el mismo punto de referencia en un 20% [16]. No duplicamos el rendimiento a pesar de duplicar el número de NIC porque el punto de referencia se convierte en CPU.

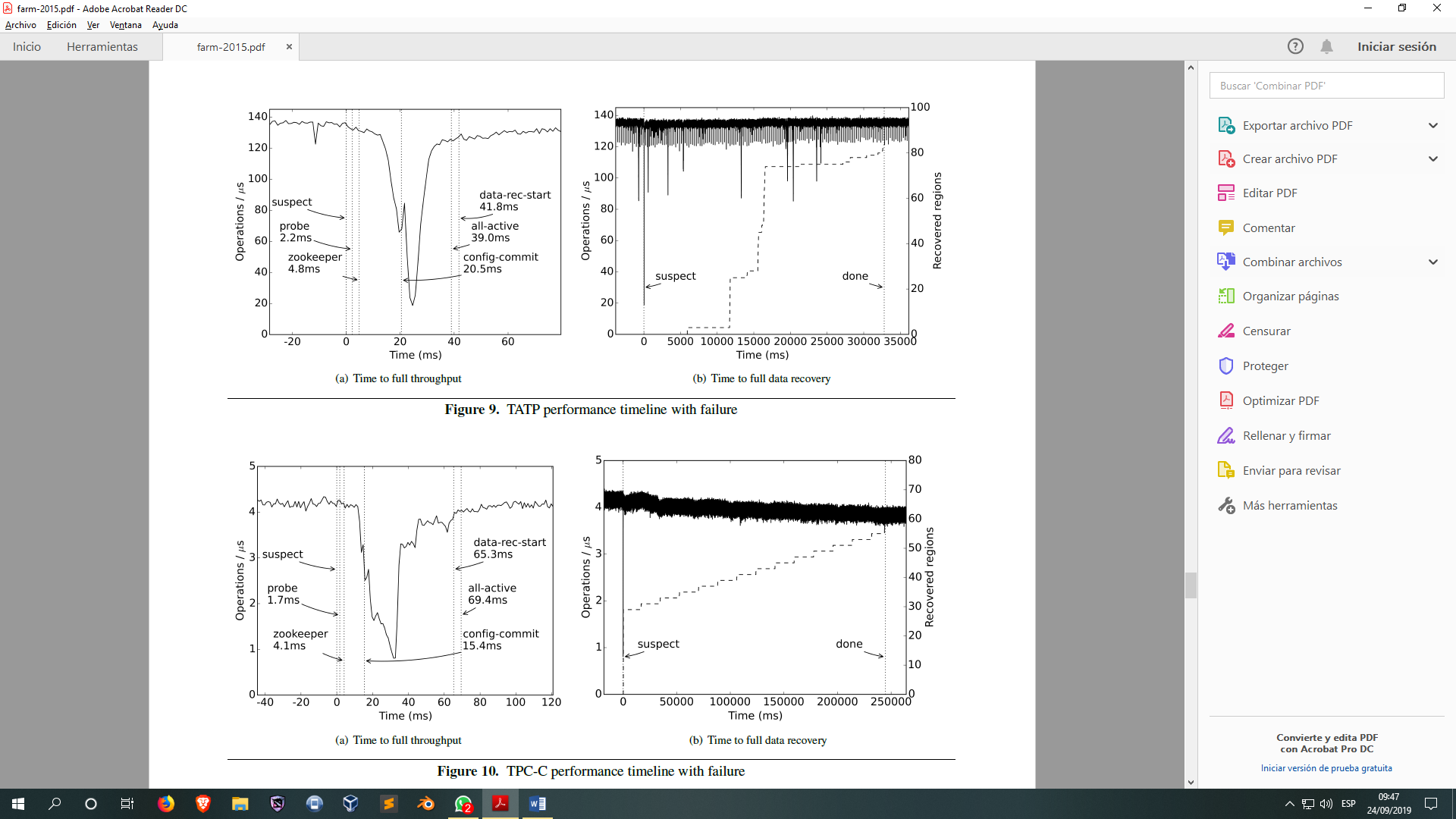
* 1. Fallas

Para evaluar el rendimiento con fallas, ejecutamos los mismos puntos de referencia y eliminamos el proceso FaRM en una de las máquinas a los 35 s del experimento. Mostramos líneas de tiempo con el rendimiento de las 89 máquinas supervivientes agregadas a intervalos de 1 ms. Las líneas de tiempo se sincronizan al inicio del experimento utilizando la mensajería RDMA. Las Figuras 9 y 10 muestran una ejecución típica de cada punto de referencia en diferentes escalas de tiempo. Ambos muestran el rendimiento como una línea continua. El "tiempo hasta el rendimiento total" es una vista ampliada sobre la falla. Muestra la hora en que expiró el contrato de arrendamiento de la máquina fallida en el CM ("sospechoso"); la hora a la que se completaron todas las sondas de lectura ("sonda"); la hora en que el CM actualizó con éxito Zookeeper ("zookeeper"); el momento en que se confirmó la nueva configuración en todas las máquinas supervivientes ("config-commit"); la hora a la que todas las regiones están activas ("todas activas"); y la hora en que comienza la recuperación de datos en segundo plano ("data-rec-start"). El "tiempo para la recuperación completa de datos" muestra una vista alejada que incluye el momento en que todos los datos se recuperan en las copias de seguridad ("hecho"). Una línea discontinua muestra el número acumulado de regiones de respaldo recuperadas con el tiempo por la recuperación de datos.

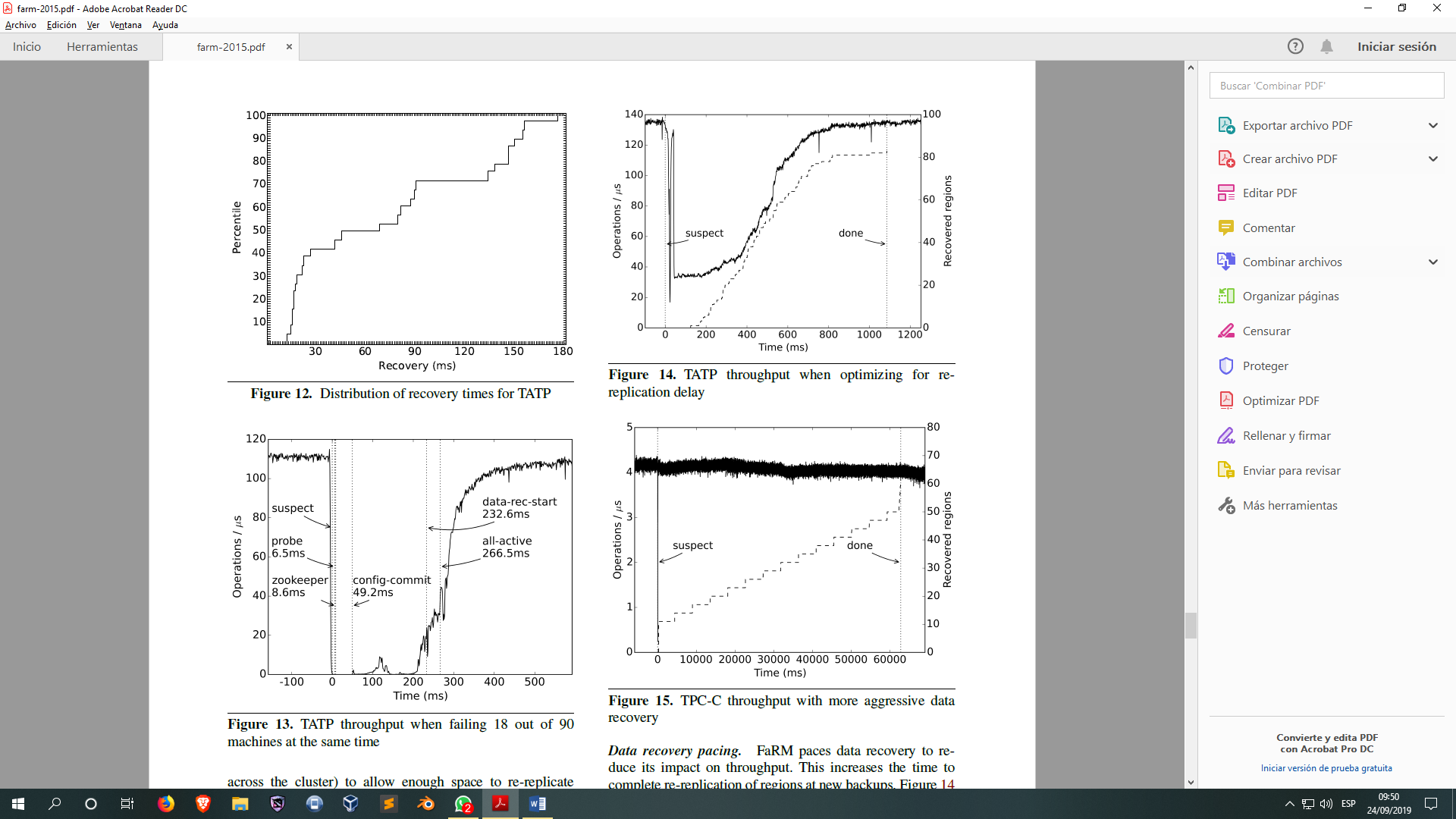
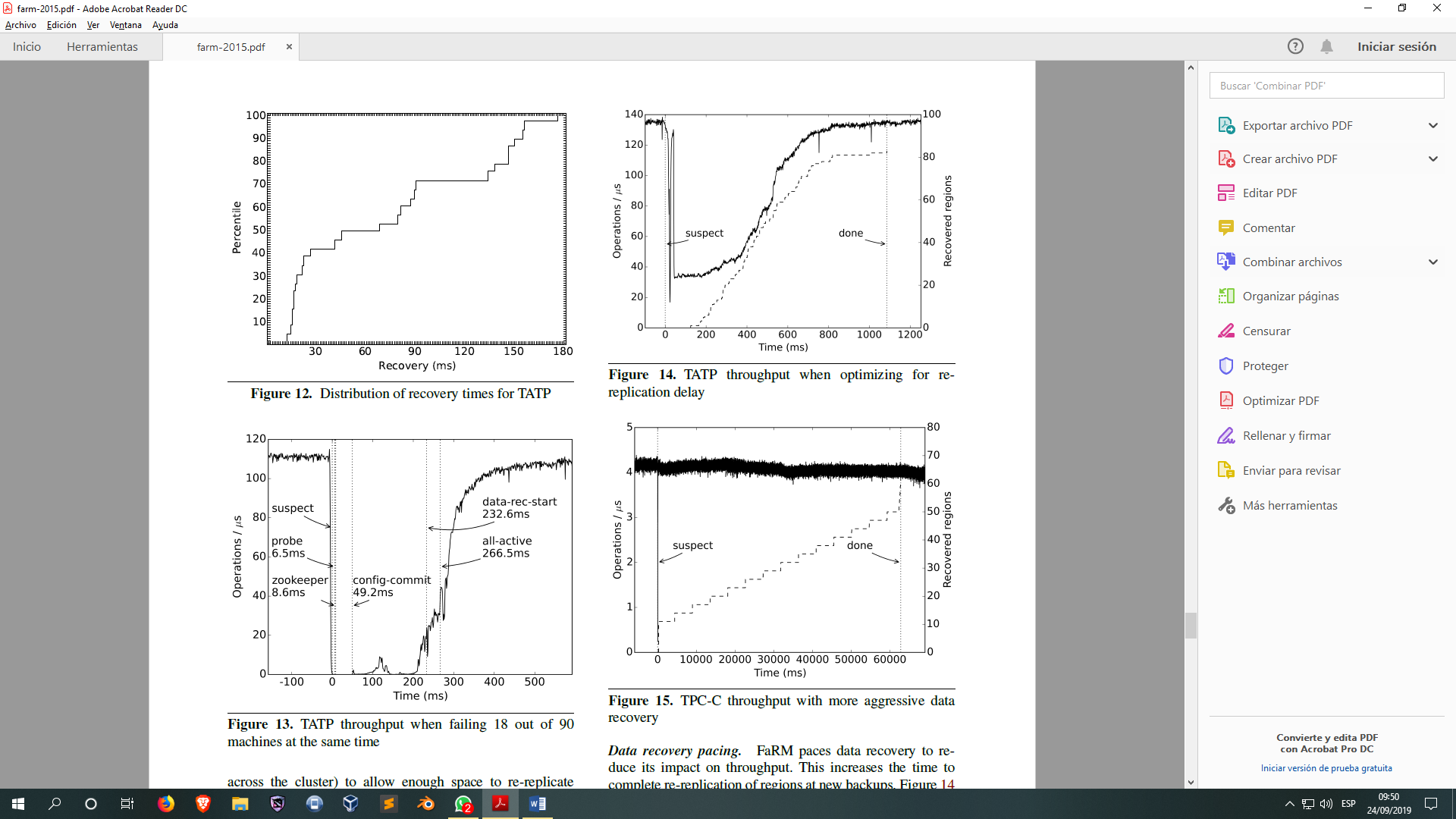
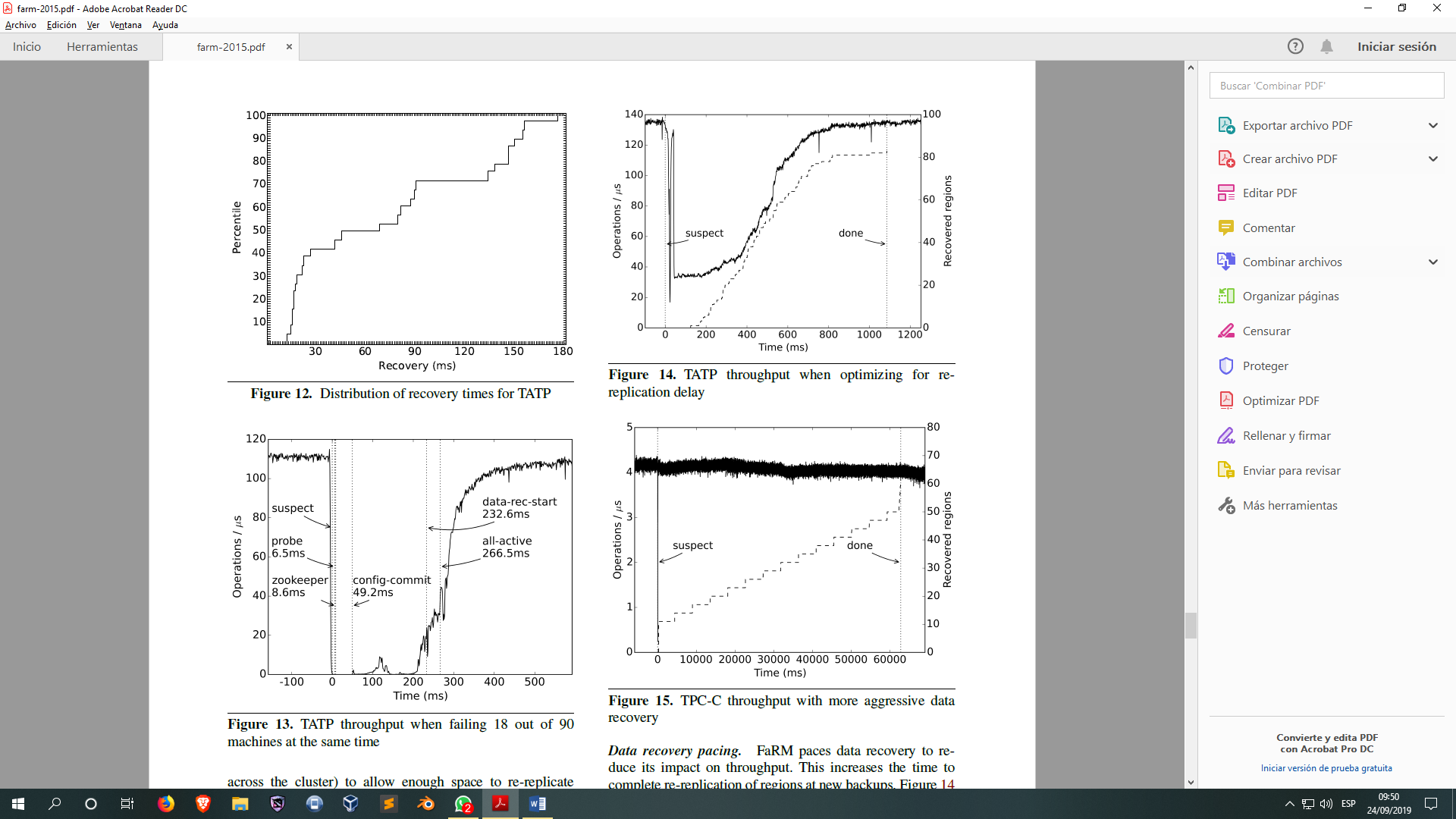
TATP. Los plazos para una ejecución típica de TATP se muestran en la Figura 9. Lo configuramos para un rendimiento máximo: cada máquina ejecuta 30 subprocesos con 8 transacciones simultáneas por subproceso. La Figura 9 (a) muestra que el rendimiento cae bruscamente en la falla pero se recupera rápidamente. El sistema vuelve al rendimiento máximo en menos de 40 ms. Todas las regiones se activan en 39 ms. La Figura 9 (b) muestra que la recuperación de datos, que es estimulada, no afecta el rendimiento en primer plano. La máquina fallida albergaba 84 regiones de 2GB. Cada subproceso obtiene bloques de 8 KB cada 2 ms, lo que significa que se necesitan alrededor de 17 s para recuperar una región de 2 GB en una sola máquina. Las máquinas recuperan una región a la vez en paralelo entre sí y aproximadamente al mismo ritmo, por lo tanto, el número de regiones recuperadas se mueve en grandes pasos. La carga de recuperación (es decir, el número de regiones por máquina que tenía una réplica en la máquina fallida) está bien equilibrada en todo el clúster: 64 máquinas recuperan una región y 10 máquinas recuperan dos. Esto explica por qué la repetición de la mayoría de las regiones se completa en alrededor de 17 segundos y por qué todas las regiones se replican nuevamente en menos de 35 segundos. Algunas regiones no están completamente asignadas, por lo que su recuperación lleva menos tiempo. Es por eso que la replicación de algunas regiones se completa en menos de 17 s. La figura también muestra que TATP tiene algunas caídas en el rendimiento incluso cuando no hay fallas. Creemos que esto se debe a un acceso sesgado en el punto de referencia; el rendimiento disminuye cuando muchas transacciones entran en conflicto y retroceden en las teclas de acceso rápido al mismo tiempo.

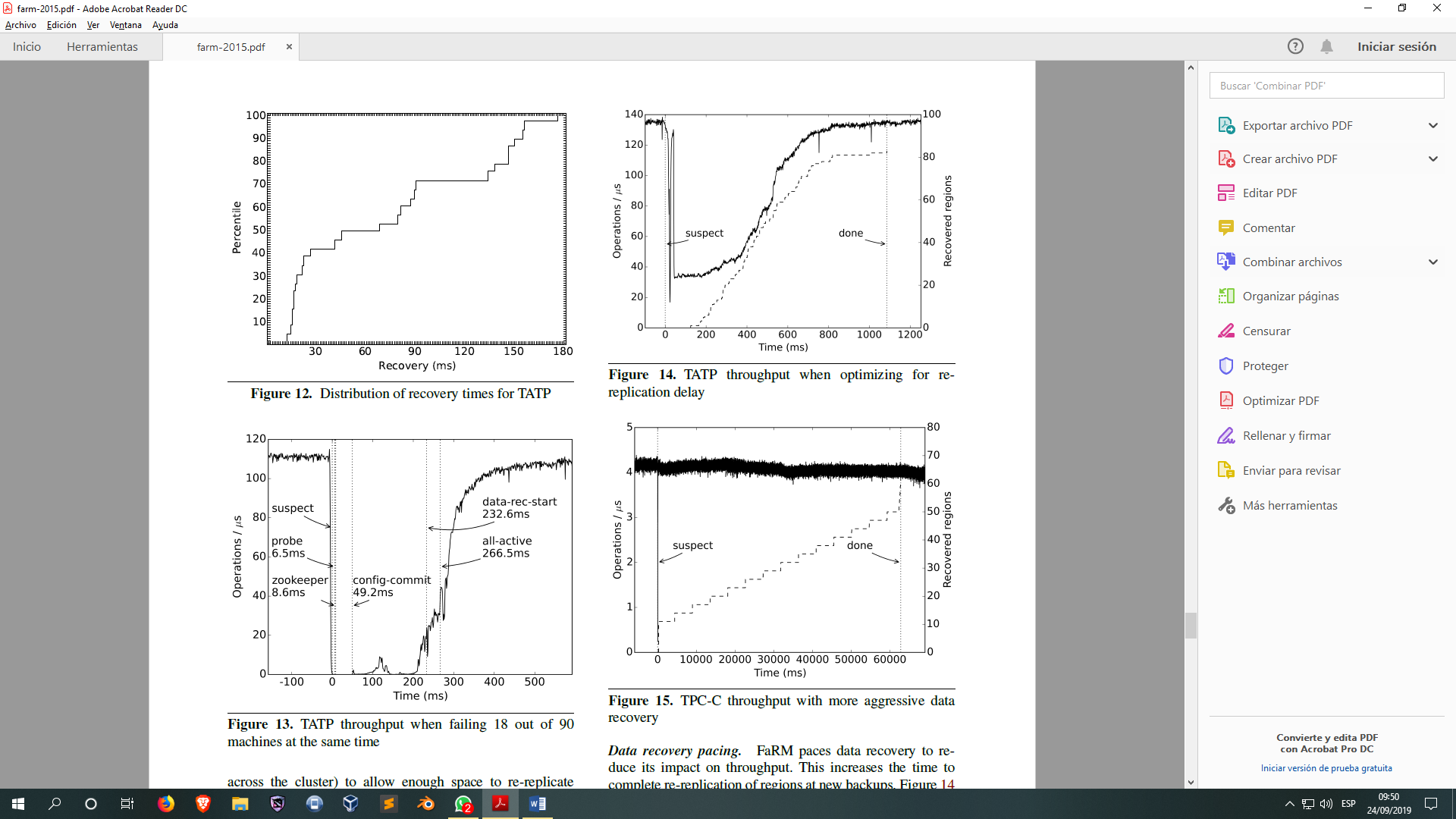
TPC-C. La Figura 10 muestra los plazos para TPC-C. La Figura 10 (a) muestra que el sistema recupera la mayor parte del rendimiento en menos de 50 ms y que todas las regiones se activan poco después. El sistema tarda un poco más de tiempo en recuperar bloqueos de transacciones que con TATP porque TPC-C tiene transacciones más complejas. La principal diferencia es que la recuperación de datos lleva más tiempo (Figura 10 (b)) a pesar de que TPCC recupera solo 63 regiones en el experimento. Esto se debe a que TPC-C reparte sus tablas hash para explotar la localidad y mejorar el rendimiento, lo que resulta en un paralelismo de recuperación reducido porque múltiples regiones se replican en el mismo conjunto de máquinas para satisfacer las restricciones de localidad especificadas por la aplicación. En el experimento, dos máquinas recuperan 17 regiones cada una, lo que lleva a la recuperación de datos en más de 4 minutos. Tenga en cuenta que el rendimiento de TPC-C se degrada gradualmente con el tiempo en la Figura 10 (b) porque el tamaño de la base de datos aumenta muy rápidamente.

Fallando el CM. La Figura 11 muestra el rendimiento de TATP a lo largo del tiempo cuando falla el proceso CM. La recuperación es más lenta que cuando falla un proceso que no es CM. Se tarda unos 110 ms para que el rendimiento vuelva al mismo nivel que antes de la falla. La razón principal del aumento en el tiempo de recuperación es un aumento en el tiempo de reconfiguración: de 20 ms en la Figura 9 (a) a 97 ms. La mayor parte de este tiempo lo dedican las nuevas estructuras de datos de construcción del CM que solo se mantienen en el CM. Debería ser posible eliminar este retraso haciendo que todas las máquinas mantengan estas estructuras de datos de forma incremental a medida que aprenden las asignaciones de regiones del CM.

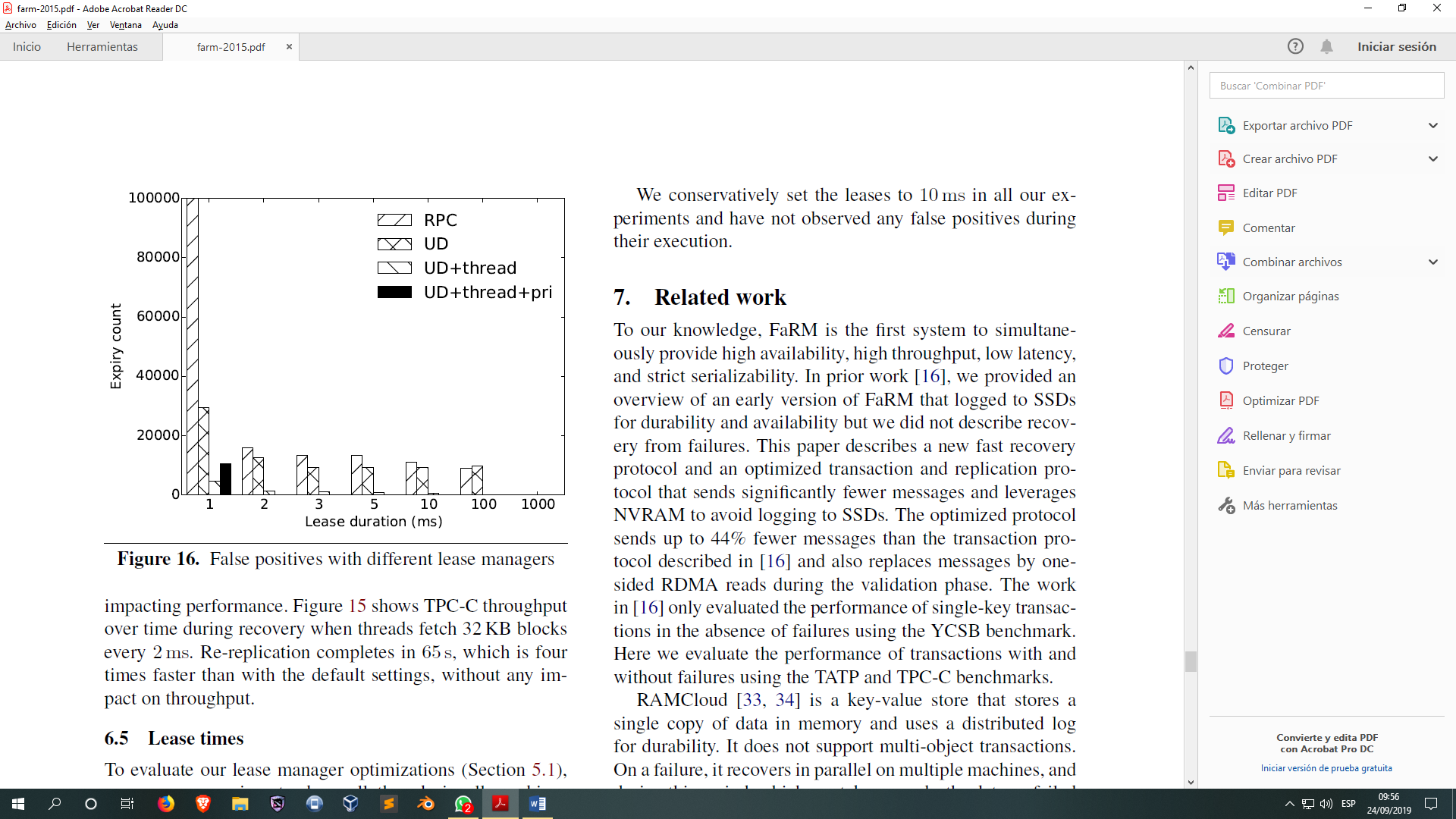
Distribución de tiempos de recuperación. Repetimos el experimento de recuperación de TATP (sin fallas de CM) 40 veces para obtener una distribución de los tiempos de recuperación. Los experimentos se ejecutaron con un conjunto de datos más pequeño (3.5 mil millones de suscriptores) para acortar los tiempos de experimento, pero confirmamos que el tiempo para recuperar el rendimiento después de una falla fue el mismo que para los conjuntos de datos más grandes. Esto se debe a que este tiempo está dominado por la recuperación del estado de la transacción, y el número de transacciones que se ejecutan simultáneamente es el mismo para ambos tamaños de conjuntos de datos. La Figura 12 muestra la distribución de los tiempos de recuperación. Medimos el tiempo de recuperación desde el punto donde el CM sospecha que la máquina falló hasta que el rendimiento se recupere al 80% del rendimiento promedio antes del fallo. El tiempo medio de recuperación es de alrededor de 50 ms y en más del 70% de las ejecuciones el tiempo de recuperación es inferior a 100 ms. En los casos restantes, la recuperación tomó más de 100 ms, pero siempre menos de 200 ms.

Fallas correlacionadas. Algunas fallas afectan a más de una máquina al mismo tiempo, por ejemplo, fallas de energía o interruptores. Para lidiar con tales fallas coordinadas, FaRM permite especificar un dominio de falla para cada máquina y el CM coloca cada réplica de una región en un dominio de falla diferente. Agrupamos máquinas en nuestro clúster en cinco dominios de falla con 18 máquinas cada uno. Esto corresponde al número de puertos en cada módulo de hoja en nuestro conmutador. Fallamos todos los procesos en uno de estos dominios de falla al mismo tiempo para simular la falla

de un interruptor de la parte superior del rack. La Figura 13 muestra el rendimiento de TATP a lo largo del tiempo para las 72 máquinas que no fallan. TATP se configuró para usar alrededor de 55 regiones en cada máquina (6.9 mil millones de suscriptores en el clúster) para permitir suficiente espacio para volver a replicar las regiones fallidas después de la falla. FaRM recupera el rendimiento máximo menos de 400 ms después de la falla. Repetimos el experimento 20 veces y esta vez fue la mediana de todos los experimentos. La mayor parte de este tiempo se dedica a recuperar transacciones. Necesitamos recuperar todas las transacciones en vuelo que modificaron cualquier región con una réplica en una máquina fallida, que leyeron una región con el primario en una máquina fallida, o que tenían el coordinador en una de las Las máquinas fallidas. Esto da como resultado

aproximadamente 130,000 transacciones que deben recuperarse, en comparación con 7500 con una sola falla. La replicación de datos demora 4 minutos porque hay 1025 regiones para replicar nuevamente. Como en experimentos anteriores, esto no afecta el rendimiento durante la recuperación debido al ritmo. Tenga en cuenta que durante este tiempo cada región todavía tiene dos réplicas disponibles, por lo que no es necesario volver a replicar de manera más agresiva.

Estimulación de recuperación de datos. FaRM estimula la recuperación de datos para reducir su impacto en el rendimiento. Esto aumenta el tiempo para completar la replicación de regiones en las nuevas copias de seguridad. La Figura 14 muestra el rendimiento a lo largo del tiempo para TATP con una recuperación de datos muy agresiva: cada subproceso obtiene cuatro bloques de 32 KB

simultáneamente. El sistema solo recupera el rendimiento máximo después de que la mayoría de las regiones se vuelven a replicar 800 ms después de la falla. Sin embargo, la recuperación de datos se completa mucho más rápido: la recuperación de 83 réplicas de región (166 GB) solo lleva 1.1 s. Utilizamos esta configuración de recuperación agresiva solo cuando las regiones pierden todas menos una réplica. La tasa de recuperación agresiva se compara favorablemente con RAMCloud [33], que recupera 35 GB en 80 máquinas en 1,6 s. TPC-C es menos sensible a la interferencia del tráfico de recuperación en segundo plano que TATP porque solo una pequeña fracción de los accesos son a objetos en máquinas remotas. Esto significa que, en configuraciones en las que es posible el ajuste específico de la aplicación, podríamos volver a replicar los datos de manera más agresiva sin afectar el rendimiento. La Figura 15 muestra el rendimiento de TPC-C a lo largo del tiempo durante la recuperación cuando los subprocesos obtienen bloques de 32 KB cada 2 ms. La nueva replicación se completa en 65 s, que es cuatro veces más rápido que con la configuración predeterminada, sin ningún impacto en el rendimiento.

* 1. tiempos de arrendamiento

Para evaluar las optimizaciones de nuestro administrador de arrendamiento (Sección 5.1), realizamos un experimento en el que todos los subprocesos de todas las máquinas emiten repetidamente lecturas de RDMA al CM durante 10 minutos. Inhabilitamos la recuperación y contamos el número de eventos de vencimiento de arrendamiento (falso positivo) en todo el clúster para diferentes implementaciones del administrador de arrendamiento y diferentes duraciones de arrendamiento. Este punto de referencia es una buena prueba de esfuerzo porque genera más tráfico en el CM que cualquiera de los puntos de referencia que describimos. La Figura 16 compara cuatro implementaciones de administrador de arrendamiento. El primero usa el RPC de FaRM (RPC). Los otros usan datagramas poco confiables: en un hilo compartido (UD), en un hilo dedicado con prioridad normal (UD + hilo), y con alta prioridad, interrupciones y sin fijación (UD + hilo + pri). Los resultados muestran que todas las optimizaciones son necesarias para permitir el uso de tiempos de arrendamiento de 10 ms o menos sin falsos positivos. Con pares de colas compartidas, incluso los arrendamientos de 100 ms caducan muy a menudo. El número de falsos positivos se reduce mediante el uso de datagramas poco confiables, pero no se elimina debido a la contención de la CPU. El uso de un subproceso dedicado nos permite usar arrendamientos de 100 ms sin falsos positivos, pero los arrendamientos de 10 ms aún caducan debido a la contención de la CPU de los procesos en segundo plano que se ejecutan en las máquinas FaRM. Con el administrador de arrendamiento interrumpido impulsado en alta prioridad, podemos usar arriendos de 5 ms durante 10 minutos sin falsos positivos. Con arrendamientos más cortos, a veces todavía tenemos falsos positivos. Estamos limitados por el tiempo de ida y vuelta de la red, que fue de hasta 1 ms con carga, y por la resolución del temporizador del sistema, que es de 0,5 ms. La resolución limitada del temporizador del sistema explica por qué el administrador de arrendamiento por interrupción tiene más falsos positivos que el basado en encuestas con arrendamientos de 1 ms. De manera conservadora, establecemos los arrendamientos a 10 ms en todos nuestros experimentos y no hemos observado ningún falso positivo durante su ejecución.

1. Conclusión

Las transacciones facilitan la programación de sistemas distribuidos, pero muchos sistemas los evitan o debilitan su consistencia para mejorar la disponibilidad y el rendimiento. FaRM es una plataforma de computación de memoria principal distribuida para centros de datos modernos que proporciona transacciones estrictamente serializables con alto rendimiento, baja latencia y alta disponibilidad. La clave para lograr esto son los nuevos protocolos de transacción, replicación y recuperación diseñados a partir de los primeros principios para aprovechar las redes de productos básicos con RDMA y un nuevo enfoque económico para proporcionar DRAM no volátil. Los resultados experimentales muestran que FaRM proporciona un rendimiento significativamente mayor y una latencia más baja que las bases de datos en memoria de última generación. FaRM también puede recuperarse de una falla de la máquina para proporcionar un rendimiento máximo en menos de 50 ms, haciendo que las fallas sean transparentes para las aplicaciones.

**Referencias**

1. Memcached. <http://memcached.org>.
2. Viking Technology. <http://www.vikingtechnology.com/>.
3. Apache Cassandra. <http://cassandra.apache.org/>, 2015.
4. MySQL. <http://www.mysql.com/>, 2015.
5. neo4j. <http://neo4j.com/>, 2015.
6. redis. <http://redis.io/>, 2015.
7. ADYA, A., DUNAGAN, J., AND WOLMAN, A. Centrifuge: Integrated lease management and partitioning for cloud services. In Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (2010), NSDI’10.
8. AGUILERA, M. K., MERCHANT, A., SHAH, M., VEITCH, A., AND KARAMANOLIS, C. Sinfonia: A new paradigm for building scalable distributed systems. In Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles (2007), SOSP’07.
9. CHANG, F., DEAN, J., GHEMAWAT, S., HSIEH, W. C., WALLACH, D. A., BURROWS, M., CHANDRA, T., FIKES, A., AND GRUBER, R. E. Bigtable: A distributed storage system for structured data. In Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (2006), OSDI’06.
10. CHOCKLER, G. V., KEIDAR, I., AND VITENBERG, R. Group communication specifications: a comprehensive study. ACM Computing Surveys (CSUR) 33, 4 (2001).
11. CORBETT, J. C., DEAN, J., EPSTEIN, M., FIKES, A., FROST, C., FURMAN, J. J., GHEMAWAT, S., GUBAREV, A., HEISER, C., HOCHSCHILD, P., HSIEH, W. C., KANTHAK, S., KOGAN, E., LI, H., LLOYD, A., MELNIK, S., MWAURA, D., NAGLE, D., QUINLAN, S., RAO, R., ROLIG, L., SAITO, Y., SZYMANIAK, M., TAYLOR, C., WANG, R., AND WOODFORD, D. Spanner: Google’s globally-distributed database. In Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (2012), OSDI’12.
12. DALESSANDRO, L., AND SCOTT, M. L. Sandboxing transactional memory. In Proceedings of the 21st ACM International Conference on Parallel Architectures and Compilation Techniques (2012), PACT’12.
13. DECANDIA, G., HASTORUN, D., JAMPANI, M., KAKULAPATI, G., LAKSHMAN, A., PILCHIN, A., SIVASUBRAMANIAN, S., VOSSHALL, P., AND VOGELS, W. Dynamo: Amazon’s highly available key-value store. In Proceedings of the the 21st ACM Symposium on Operating Systems Principles (2007), SOSP’07.
14. DIACONU, C., FREEDMAN, C., ISMERT, E., LARSON, P.-°A ., MITTAL, P., STONECIPHER, R., VERMA, N., AND ZWILLING, M. Hekaton: SQL Server’s memory-optimized OLTP engine. In Proceedings of the ACM SIGMOD International Conference on Management of Data (2013), SIGMOD’ 13.
15. DICE, D., SHALEV, O., AND SHAVIT, N. Transactional locking II. In Proceedings of the 20th International Symposium on Distributed Computing (2006), DISC’06.
16. DRAGOJEVI´C, A., NARAYANAN, D., HODSON, O., AND CASTRO, M. FaRM: Fast remote memory. In Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (2014), NSDI’14.
17. GRAEFE, G. Write-optimized B-trees. In Proceedings of the 30th International Conference on Very Large Data Bases (2004), VLDB’04. 69
18. GRAY, C., AND CHERITON, D. Leases: An efficient faulttolerant mechanism for distributed file cache consistency. SIGOPS Operating Systems Review (OSR) 23, 5 (1989).
19. GRAY, J., AND REUTER, A. Transaction Processing: Concepts and Techniques. 1992.
20. GUERRAOUI, R., AND KAPALKA, M. On the correctness of transactional memory. In Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (2008), PPoPP’08.
21. HUNT, P., KONAR, M., JUNQUEIRA, F. P., AND REED, B. Zookeeper: wait-free coordination for internet-scale systems. In Proceedings of the 2010 USENIX Annual Technical Conference (2010), USENIX ATC’10.
22. INFINIBAND TRADE ASSOCIATION. Supplement to Infini-Band Architecture Specification Volume 1 Release 1.2.2 Annex A16: RDMA over Converged Ethernet (RoCE), 2010.
23. KALIA, A., KAMINSKY, M., AND ANDERSEN, D. G. Using RDMA efficiently for key-value services. In Proceedings of the 2014 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (2014), SIGCOMM’14.
24. LAMPORT, L. The part-time parliament. ACM Transactions on Computer Systems 16, 2.
25. LAMPORT, L., MALKHI, D., AND ZHOU, L. Vertical Paxos and primary-backup replication. In Proceedings of the 28th ACM Symposium on Principles of Distributed Computing (2009), PODC’09.
26. LARSON, P.-°A., BLANAS, S., DIACONU, C., FREEDMAN, C., PATEL, J. M., AND ZWILLING, M. High-performance concurrency control mechanisms for main-memory databases. PVLDB 5, 4 (2011).
27. LEHMAN, P. L., AND YAO, S. B. Efficient locking for concurrent operations on B-trees. ACM Transactions on Database Systems 6, 4 (Dec. 1981).
28. MICROSOFT. Scaling out SQL Server. <http://www.microsoft.com/en-us/servercloud/solutions/high-availability.aspx>.
29. MICROSOFT. Open CloudServer OCS V2 specification:Blade, 2014.
30. MICROSOFT. OCS Open CloudServer power supply v2.0. <http://www.opencompute.org/wiki/Server/SpecsAndDesigns>, 2015.
31. MITCHELL, C., YIFENG, G., AND JINYANG, L. Using onesided RDMA reads to build a fast, CPU-efficient key-value store. In Proceedings of the 2013 USENIX Annual Technical Conference (2013), USENIX ATC’13.
32. NEUVONEN, S., WOLSKI, A., MANNER, M., AND RAATIKKA, V. Telecom Application Transaction Processing benchmark. <http://tatpbenchmark.sourceforge.net/>.
33. ONGARO, D., RUMBLE, S. M., STUTSMAN, R., OUSTERHOUT, J., AND ROSENBLUM, M. Fast crash recovery in RAMCloud. In Proceedings of the 23rd ACM Symposium on Operating Systems Principles (2011), SOSP’11.
34. RUMBLE, S. M., KEJRIWAL, A., AND OUSTERHOUT, J. Log-structured Memory for DRAM-based Storage. In Proceedings of the 12th USENIX Conference on File and Storage Technologies (2014), FAST’14.
35. SETHI, R. Useless actions make a difference: Strict serializability of database updates. JACM 29, 2 (1982).
36. SHAUN HARRIS. Microsoft reinvents datacenter power backup with new Open Compute project specification. <http://blogs.msdn.com/b/windowsazure/archive/2012/11/13/windows-azurebenchmarks-show-top-performance-forbig-compute.aspx>, 2015.
37. SOWELL, B., GOLAB, W. M., AND SHAH, M. A. Minuet: A scalable distributed multiversion B-tree. PVLDB 5, 9 (2012).
38. TRANSACTION PROCESSING PERFORMANCE COUNCIL (TPC). TPC benchmark C: Standard specification. <http://www.tpc.org>.
39. TU, S., ZHENG, W., KOHLER, E., LISKOV, B., AND MADDEN, S. Speedy transactions in multicore in-memory databases. In Proceedings of the 24th Symposium on Operating Systems Principles (2013), SOSP’13.
40. ZHENG, W., TU, S., KOHLER, E., AND LISKOV, B. Fast databases with fast durability and recovery through multicore parallelism. In Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (2014), OSDI’14.