SISTEMAS LÍQUIDOS, SOLUCIÓN A LOS PROBLEMAS DE SISTEMAS CONVENCIONALES

María Paula Riveros Gómez
Estudiante de pregrado
Escuela de Ingeniería de sistemas e Informática
Universidad Industrial de Santander
Bucaramanga, Colombia
paulariveros0321@gmail.com

Miguel Ángel Duarte Delgado
Estudiante de pregrado
Escuela de Ingeniería de sistemas e Informática
Universidad Industrial de Santander
Bucaramanga, Colombia
angel4dvc@gmail.com

Elkin Darío Fernández Celis
Estudiante de pregrado
Escuela de Ingeniería de sistemas e Informática
Universidad Industrial de Santander
Bucaramanga, Colombia
elkinfernandez21@gmail.com

Abstract

Distributed systems require a system to protect stored data against any failure in them and recover them in case of loss, however this system may change when the request for access to information exceeds the speed of data recovery. Now, a "liquid system" shows, based on how your data repair policy works, that instead of reacting and repairing separately lost data fragments of each failed node, the lost data fragments of many failed nodes are grouped together to form a "liquid" and repair as a fluid. This represents an attractive solution to various difficulties of specific systems. Detailed evidence that the implementation of this new approach greatly modified the deficiencies of the systems currently used.

Keywords - bottleneck, fragments, liquid system, node, objects.

Resumen

Los sistemas distribuidos poseen un sistema para proteger los datos almacenados contra cualquier falla en estos y recuperarlos en caso de pérdida, sin embargo, este sistema puede alterarse cuando la solicitud de acceso a la información supera la velocidad de recuperación de los datos. Ahora, un "sistema líquido" muestra, basado en cómo funciona su política de reparación de datos que, en lugar de reaccionar y reparar por separado fragmentos de datos perdidos de cada nodo fallado, los fragmentos perdidos de datos de muchos nodos fallidos se agrupan para formar un "líquido" y se reparan como un fluido. Esto representa una atractiva solución a dichas dificultades de los sistemas convencionales. Las pruebas muestran que la implementación de este nuevo enfoque de sistema distribuido disminuye en gran medida las deficiencias de los sistemas usados actualmente.

Palabras clave - cuello de botella, fragmentos, sistema líquido, nodo, objetos

I. INTRODUCCIÓN

A través de los años, en el campo de la computación, las personas han buscado la innovación, creación, solución a problemas para hacer más eficientes sus necesidades y tener un mejor resultado de lo que se espera; tal es el caso de los sistemas distribuidos de almacenamiento, una colección de computadores independientes que aparentan ejecutarse como un único computador, es decir, múltiples computadores hacen la tarea que al final tienen un único objetivo, almacenar datos de manera duradera. La información es almacenada en nodos, los cuales pueden tener fallos, de estos distinguimos dos, los fallos en el software del que pueden recuperarse y los fallos en el hardware donde no se puede recuperar lo que se perdió; entonces se ejecuta una rápida reparación lo que exige un gran ancho de banda de red y por ende se genera a lo que en computación se conoce como cuello de botella, donde ocurre un aumento del tráfico de la red necesario para reparar los datos perdidos debido a los nodos que fallaron. Por este motivo aparecen los sistemas líquidos, que proporcionan medios más confiables y duraderos de almacenamiento y hacen frente al cuello de botella, siendo más eficientes y eficaces a la hora de almacenar información y cuando se requiere de esta de manera rápida.

II. ESTADO DEL ARTE

Una aplicación de este tema puede ser Windows Azure Storage (WAS) [1], un sistema de almacenamiento en la nube que brinda a los clientes la capacidad de almacenar cantidades aparentemente ilimitadas de datos durante cualquier período de tiempo. Los clientes de WAS tienen acceso a sus datos desde cualquier lugar en cualquier momento y solo pagan por lo que usan y almacenan. En WAS, los datos se almacenan de manera duradera logrando una replicación local y geográfica para facilitar la recuperación ante desastres; esto a partir de un conjunto de códigos para la codificación de borrado llamado Códigos de reconstrucción local (LRC).

LRC reduce la cantidad de fragmentos de codificación de borrado que deben leerse cuando se reconstruyen fragmentos de datos que están fuera de línea, mientras se mantiene baja la sobrecarga de almacenamiento. Los beneficios importantes de LRC son que reduce el ancho de banda y las E / S necesarias para las lecturas de reparación sobre los códigos anteriores, al tiempo que permite una reducción significativa en la sobrecarga de almacenamiento [2]. De esta manera se usa el LRC en WAS para proporcionar almacenamiento duradero de baja sobrecarga con latencias de lectura consistentemente bajas. Actualmente, el almacenamiento WAS viene en forma de blobs (archivos), tablas (almacenamiento estructurado) y colas (entrega de mensajes).

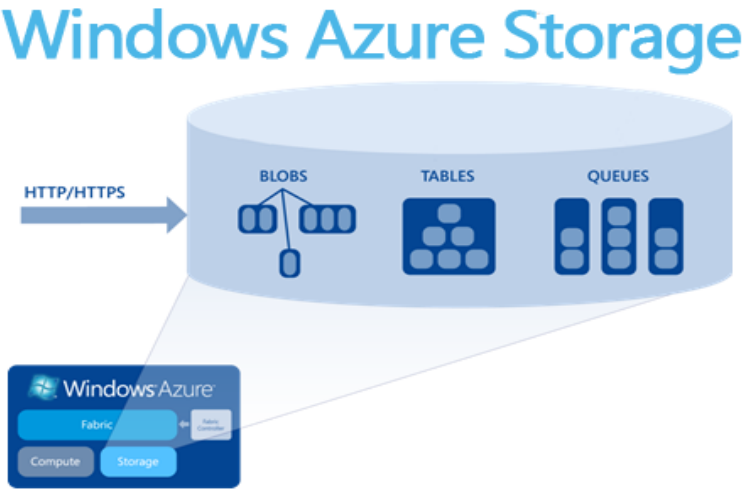

Figura 1. Windows Azure Storage.

III. MARCO TEÓRICO

Comenzando por lo básico, el almacenamiento en la nube consiste en guardar archivos en un lugar de internet. Esos lugares de internet son aplicaciones o servicios que almacenan o guardan esos archivos; los archivos pasan de estar en nuestros dispositivos a estar guardados en ese servicio o aplicación. Pero la Nube no es algo tan abstracto y efímero como lo trasmite su definición; las palabras Nube o Cloud son más unos términos de marketing que sirven para describir aquellos servicios de internet que hacen algo más que mostrar páginas web, por ejemplo, guardar archivos o un programa de contabilidad. Son programas o servicios que no están en tu ordenador, están en internet, por eso se dice que están en la Nube. Sin embargo, la Nube son en realidad dispositivos físicos, ordenadores (servidores) conectados a internet con discos enormes que pueden guardar archivos.

Dado que la mira central de este análisis estará puesta en la observación, exploración y argumentación de un naciente concepto relacionado con los sistemas de almacenamiento de objetos distribuidos, será preciso plantear y aclarar algunos conceptos centrales. Se entiende como sistema de almacenamiento distribuido a aquel que se compone de una gran cantidad de nodos de almacenamiento interconectados, con cada nodo capaz de almacenar una gran cantidad de datos; datos que son almacenados en redes de computadores inalámbricas que son conocidas como redes en la nube. Los objetivos clave de un sistema de almacenamiento distribuido son almacenar la mayor cantidad posible de datos de origen, minimizar el tiempo de acceso a los datos de origen por parte de los usuarios o aplicaciones y asegurar un alto nivel de durabilidad de los datos de origen; Para lograr esta última ante fallas en los componentes, se utiliza una fracción de la capacidad de almacenamiento sin procesar para almacenar datos redundantes.

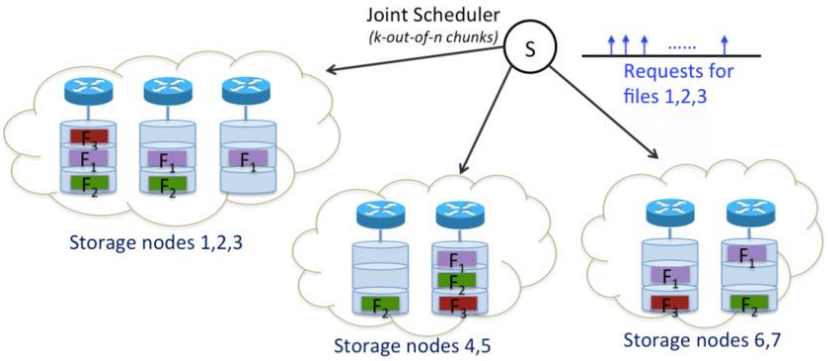

Figura 2. Sistema de almacenamiento distribuido equipado con 7 nodos y que almacena diferentes códigos de borrado.

Ahora pues, estos sistemas de almacenamiento han presentado diferentes características que afectan negativamente el manejo de los datos almacenados. De manera que, estudiosos como Michael G. Luby [4], entre otros, han intentado replantear ideas hasta llegar a lo que denominan Liquid Cloud Storage, o traducido en español, almacenamiento líquido en la nube, el cual es un concepto que surge del deseo de solventar aquellas características negativas, o mejor, dificultades presentes en los sistemas convencionales; para ser más precisos, la necesidad de establecer medios más confiables y duraderos de almacenamiento y para hacer frente a lo que en computación se conoce como cuello de botella en donde se entorpece, dificulta y/o se estanca el flujo normal de datos a través de la red y que es causado por la ejecución de códigos dedicados a realizar recuperaciones de datos. Al haber pérdida de datos se ejecutan los códigos para realizar las respectivas recuperaciones de datos, que son necesitados de manera urgente, de tal forma que se ejecuta una rápida reparación lo que exige un gran ancho de banda de red para recuperarlos generando dicho cuello de botella por la cantidad de datos a recuperar.

Por otra parte, los sistemas líquidos son una nueva clase de sistemas de almacenamiento distribuido los cuales utilizan códigos grandes para distribuir los datos almacenados para cada objeto en una gran cantidad de nodos, y utilizan una estrategia de reparación perezosa (lazy repair strategy) para reparar lentamente la perdida de datos en los nodos que han presentado falla. Este término, sistema líquido, está inspirado en cómo funciona la política de reparación puesto que, en lugar de reaccionar y reparar por separado fragmentos de datos perdidos de cada nodo fallado, los fragmentos perdidos de datos de muchos nodos fallidos se agrupan para formar un "líquido" y se reparan como un fluido.

De lo anterior mencionado, podemos formar una base concisa para proseguir develando más conceptos y características relacionadas con este nuevo modo de ver los sistemas de almacenamiento distribuido y extender la búsqueda e indagación en asuntos relacionados con el tema que trataremos.

IV. SISTEMAS DE CODIGO PEQUEÑO (SMALL CODE SYSTEM)

Usan el sistema de replicación donde cada fragmento es una copia del objeto original, ejemplificado de un código de borrado trivial MDS. La triplicación es un simple código de borrado MDS en el que el objeto se puede recuperar de una de cualquiera de las tres copias, en general muchos sistemas de almacenamiento distribuido usan la replicación. Un código RS (Reed-Solomon) es un código MDS usado en una variedad de aplicaciones, muy popular en sistemas de almacenamiento.

La reparación reactiva se usa para sistemas de código pequeño, donde el reparador regenera rápidamente fragmentos tan pronto como se pierden de un nodo, de tal manera que lee los k fragmentos disponibles para reparar el fragmento faltante del objeto. A causa de perder un fragmento por un objeto debido a una falla de nodo, la probabilidad de falla de nodos adicionales en un corto intervalo de tiempo cuando el r es lo suficientemente significativo como para que la operación deba completarse tan rápido como sea practico. En general, la reparación reactiva utiliza grandes ráfagas de tráfico de reparación durante cortos periodos de tiempo

para reparar objetos lo más rápido y seguro posible después de que se declare que un nodo que almacena datos para los objetos ha fallado de manera permanente.

V. SISTEMAS LIQUIDOS

Usan una combinación de códigos grandes *(n; k; r)*, reparación diferida y organización del almacenamiento en flujo. El código RaptorQ es un código de borrado adecuado para un sistema líquido, están diseñados fundamentalmente para admitir grandes valores de la tupla *(n; k; r)* con baja complejidad de codificación y decodificación y tener propiedades de recuperación excepcionales. Es un código fuente, que puede generar símbolos codificados *n* como se desee para un valor dado de *k.*

Para un sistema líquido, el valor de *r* es grande, por esta razón se puede una la reparación diferida para recuperar los fragmentos perdidos con un ancho de banda reducido. De esta manera se ve que se puede usar un sistema líquido para combinaciones flexibles y optimas de durabilidad de almacenamiento, sobrecarga de almacenamiento, uso de ancho de banda de reparación y rendimiento en el acceso de los datos, donde este último supera al de los sistemas de código pequeño.

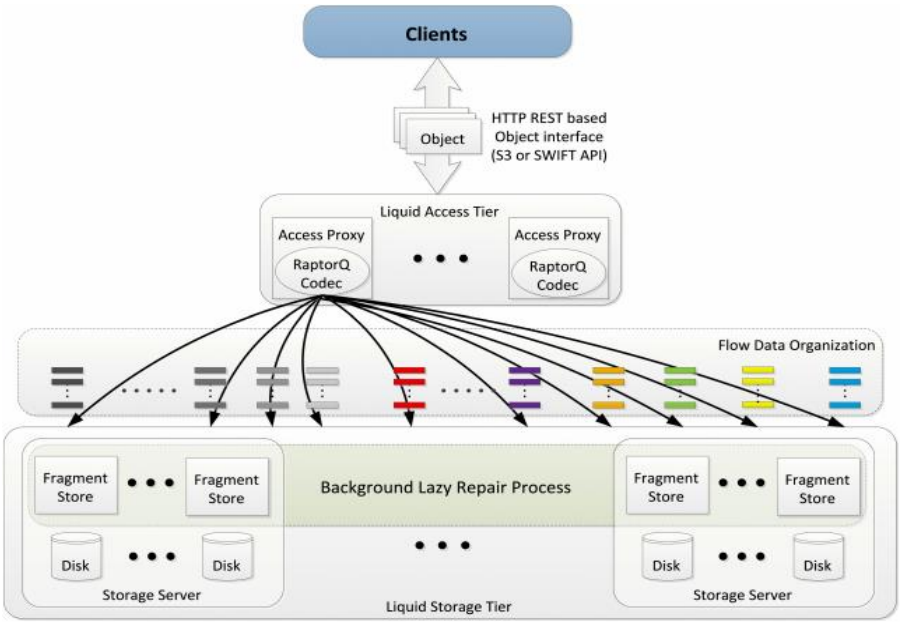

Figura 3. Arquitectura de un sistema líquido.

La Figura 3 muestra un posible constructo de lo que sería una arquitectura de sistema líquido, donde los usuarios usan la interfaz estándar, como la API S3 o SWIFT, para realizar solicitudes de acceso a objetos y almacenamiento, el almacenamiento de estos objetos y las solicitudes de acceso son distribuidos a través de servidores proxy de acceso en el nivel líquido. Los proxys usan códec RaptorQ para codificar objetos en fragmentos y decodificar fragmentos en objetos, según la organización del almacenamiento de flujo. Se accede a los servidores de proxy para leer y escribir fragmentos desde y hacia los servidores de almacenamiento y así almacenar los fragmentos en los discos dentro del nivel de almacenamiento líquido. De este modo, la reparación diferida funciona como un proceso de segundo plano dentro del nivel de almacenamiento líquido [3].

Ventajas de los Sistemas Líquidos

- Resuelven el problema de cuello de botella con menos sobrecarga de almacenamiento y un gran MTTDL (*tiempo medio hasta la perdida de cualquier fuente de datos*). La tasa de reparación promedio de los sistemas líquidos es menor que la de los códigos pequeños, y su taza de reparación máxima es sustancialmente menor que la de los sistemas de códigos pequeños, para una misma sobrecarga de almacenamiento.

- Pueden ser flexiblemente desplegados para diferentes gastos de almacenamiento y reparar tráfico usando puntos de funcionamiento.
- Evitan que se presente un fallo de reparación en un nodo transitorio, lo que era inevitable en sistemas de códigos pequeños.
- El sistema líquido de reparación regulada proporciona un gran MTTDL incluso cuando los errores de nodos son esencialmente contradictorios.
- En un prototipo de Sistema Líquido se demostró la gran velocidad de acceso que tienen los sistemas líquidos contra los sistemas de código pequeño.

VI. FALLOS DEL SISTEMA

Las fallas son comunes e inevitables en un sistema de almacenamiento distribuido y si se usa hardware básico se aumenta la frecuencia de fallas, lo mismo con las operaciones de mantenimiento y servicio necesarias, así como las actualizaciones de software o la reconfiguración del sistema a nivel de hardware o software. Si la falla es a nivel hardware, por ejemplo, un disco duro puede fallar a nivel de sectores y estos serán ilegibles donde puede ser algo permanente, pero será transitoria si por ejemplo retomando de nuevo el ejemplo del disco duro, esta vez se han generado errores a nivel software (un corte, reinicio o cambio de software).

Los componentes de los sistemas de almacenamiento distribuidos se pueden actualizar y reemplazar. Un modelo de algún dispositivo puede tener diferentes ejecuciones de fabricación que permite variaciones en la vida útil del dispositivo, además las actualizaciones que estas requieran y cambios como la transición de discos duros a los SSD, también afectan las estadísticas y la naturaleza de las fallas de almacenamiento.

El comportamiento de falla en un sistema de almacenamiento distribuido es complicado de modelar, los modelos de falla se enfocan en algo bastante sencillo, que son: las fallas de nodo y fallas de sector.

Fallas de nodo: Su análisis proporciona información sobre las fortalezas y debilidades de un sistema practico y una aproximación de como funcionara dicho sistema. Hay dos tipos de fallas en los nodos: la falla de nodo transitorio, cuando los nodos dejan de responder por cierto tiempo y los datos que almacenan no están disponible durante ese tiempo; ese tiempo es desconocido y varia en un rango que puede durar desde segundo hasta incluso meses. Y la falla de nodo, en donde los datos guardados en dichos nodos no están disponibles permanentemente.

Fallas del sector: Es un tipo de pérdida de datos donde pueden ocurrir problemas operativos. Este sucede cuando un sector de datos que está almacenado en un nodo se degrada y no se puede leer, donde la perdida de datos solamente se detecta cuando se intenta leer el sector desde el nodo. La depuración de datos se usa muy a menudo para detectar fallos en el sector en el que se intenta leer todos los datos almacenados en el sistema con cierta frecuencia, ej.: Dropbox elimina datos con la frecuencia de una vez cada 2 semanas e informa que el tráfico de lectura debido a la eliminación puede ser mayor que el resto del tráfico de datos de lectura combinado. La mayoría de los proveedores de sistemas de almacenamiento de datos eliminan los datos, pero no informan la frecuencia de estos, sin embargo, la limpieza puede tener un impacto en el rendimiento del sistema.

VII. REPARACIÓN

La codificación de borrado es muy eficiente debido a que reduce los gastos generales de almacenamiento y mejora la durabilidad, esto es bueno pero requiere gran cantidad de ancho de banda para reparar la pérdida de los fragmentos por causa de las fallas de nodos, y esto puede generar “cuello de botella de reparación”; el objetivo de los autores del artículo de la arquitectura de sistemas líquidos es poder diseñar esquemas de codificación que permitan codificar los datos sin tener el problema del cuello de botella de reparación, esto permitirá ahorrar petabytes de datos de gastos en almacenamiento y reduciría los costos de cluster [3].

Para mostrar esto más claramente, nos referimos a que al ser el cuello de botella un obstáculo para pasar de la replicación a un código de borrado más eficiente como el RS, se genera el problema de que al usar RS para recuperar un fragmento debido al fallo en un nodo se necesita que se transfiera información de k fragmentos para su recuperación, lo que aumenta el ancho de banda requerido para su recuperación.

El problema de cuello de botella ha llevado a la búsqueda de códigos que brinden localidad, es decir que la reparación requiera menor transferencia de *k* fragmentos para su reparación, por lo tanto, se introducen los códigos LR y se proporcionan compensaciones con respecto a la localidad y otros parámetros del código.

VIII. REPARACIÓN EN SISTEMAS DE CÓDIGO PEQUEÑO Y EN SISTEMAS LIQUIDOS

Con respecto al análisis planteado por los autores se realizó una tabla para comparar como es la reparación en cada sistema.

R_{peak}: Valor que determina la infraestructura de red a usar para soportar la reparación del sistema, en general se establece un valor grande de R_{peak} de limite que sea superior a la tasa de reparación de lectura y de esto se halla su MTTDL, que corresponde al tiempo logrado para el sistema con el valor de R_{peak} dado

Sistemas de Código Pequeño	**Sistemas Líquidos**
Usan reparación reactiva; $\boldsymbol{R_{peak}}$ se establece en un valor más alto que el ancho de banda de reparación promedio, lo que provoca una ráfaga de reparación cuando ocurre una falla del nodo y los fragmentos se reparan lo más rápido posible. Solo una pequeña parte de objetos está en cola de reparación en cualquier momento.	Usan una reparación diferida; y usa un ancho de banda de reparación promedio sustancialmente menor a la reparación reactiva de los sistemas de códigos pequeños. El valor del R_{peak} es establecido siendo lo suficientemente bajo para que la cola de reparación contenga casi todos los objetos todo el tiempo.
El valor de $\boldsymbol{R_{peak}}$ se tiene que asignar manualmente.	Los valores de R_{peak} se calculan algorítmicamente para sistemas líquidos.
El $\boldsymbol{T_{RIT}}$ debe ser relativamente pequeño, ya que el ancho de banda usado es muy grande.	El T_{RIT} usa valores grandes, lo que tiene el beneficio de eliminar prácticamente la reparación innecesaria debido a fallas transitorias.
Es un desafío encontrar un equilibrio entre usar pocos grupos de ubicación, lo que significa que hay gran cantidad de datos de origen en el grupo de ubicación y usar muchos grupos de ubicación, que implica que hay muchas maneras en las que pueda ocurrir la pérdida de datos de origen, ambas degradan el MTTDL.	Usa un solo grupo de ubicación para todos los objetos, por lo tanto, el número de nodos debe ser mucho más grande para que el objeto se pierda. La estructura de objetos anidados implica que, si el objeto con el menor número de fragmentos se recupera, entonces todos los objetos pueden recuperarse.
Presentan vulnerabilidad, ya que al perder un pequeño número de fragmentos puede conducir a la perdida de objetos.	Son más resistentes a la vulnerabilidad presentada en los códigos pequeños ya que para que un objeto se pierda, se tiene que perder la mayor cantidad de fragmentos posible.
Requieren una gran cantidad de ancho de banda para lograr un MTTDL razonable, lo que a veces no se logra debido a las limitaciones de ancho de banda para realizar una operación de gran valor.	Se usa un ancho de banda de reparación promedio sustancialmente menor y un ancho de banda de reparación pico dramáticamente menor que un sistema de código pequeño.

Tabla 1. Diferencias entre sistemas de código pequeño y sistemas líquidos.

IX. IMPLEMENTACION DE PROTOTIPO

Se implementó un prototipo del sistema líquido para los siguientes objetivos: Comprender y mejorar las características operativas de la implementación de tal sistema en un entorno del mundo real, comparar el rendimiento de acceso de los sistemas líquidos y los sistemas de código pequeño, verificar en forma cruzada el simulador de reparación diferida que usa una tasa de reparación de lectura fija para asegurar que ambos produzcan el mismo MTTDL en las mismas condiciones, y por último, validar el comportamiento básico de una tasa de reparación de lectura regulada. El hardware utilizado en el prototipo consiste en un rack de 14

servidores. Los servidores están conectados a través de enlaces Ethernet dúplex completo de 10 Gbit a un conmutador y están equipados con CPU Intel Xeon que funcionan a 2,8 GHz, y cada CPU tiene 20 núcleos. Cada servidor está equipado con una unidad SSD de 600 GB de capacidad que se utiliza para el almacenamiento.

El software del prototipo de sistema líquido está escrito a medida en el lenguaje de programación Go. Consta de cuatro módulos principales:

- Software de nodo de almacenamiento. Es un servidor HTTP que se utiliza para almacenar fragmentos en la unidad SSD. Dado que un sistema líquido generalmente usaría muchos más de 14 nodos de almacenamiento, se ejecutaron muchas instancias del software del nodo de almacenamiento en una pequeña cantidad de servidores físicos, emulando así sistemas con cientos de nodos.
- Un generador de acceso crea solicitudes de usuario aleatorias para los datos del usuario y los envía a los accesores. También recopila los tiempos de acceso resultantes de los accesores.
- Los accesores toman las solicitudes de datos del usuario del generador de acceso y crean las solicitudes correspondientes para fragmentos (completos o parciales) de los nodos de almacenamiento. Reúnen las respuestas de fragmentos y miden el tiempo que lleva hasta que se hayan recibido suficientes fragmentos para recrear los datos del usuario. En general, hay múltiples accesos ejecutándose en el sistema.
- Se utiliza un proceso de reparación para gestionar la cola de reparación y regenerar fragmentos faltantes de objetos almacenados. El proceso de reparación se puede configurar para usar una velocidad de reparación de lectura fija, o para usar una velocidad de reparación de lectura regulada.

Configuración de rendimiento de acceso

Los usuarios del sistema son modelados por el módulo generador de acceso. Se establece una carga de acceso de usuario objetivo $\rho(0 < \rho < 1)$ en el sistema de la siguiente manera. Sea C la capacidad agregada de los enlaces de red a los nodos de almacenamiento, y suponga s el tamaño de las solicitudes de datos del usuario que se emitirán. Entonces, las $C = s$ solicitudes por unidad de tiempo usarían toda la capacidad disponible. Se establece el tiempo medio entre solicitudes de usuario sucesivas en $t = s/(\rho \cdot C)$, de modo que, en promedio, se usa una fracción ρ de la capacidad. El módulo generador de acceso utiliza una variable aleatoria exponencial para generar el tiempo de cada solicitud de datos de usuario sucesiva, donde la media de la variable aleatoria es t. Como ejemplo, hay un enlace de 10 Gbps a cada uno de los seis nodos de almacenamiento en la configuración de la Figura 4, y por lo tanto $C = 60$ Gbps. Si las solicitudes de datos del usuario son de tamaño $s = 10$ MB $= 80 \cdot 2^{20}$ bits, y la carga objetivo es $\rho = 0.8$ (80% de la capacidad), el tiempo medio entre solicitudes es:

$$t = \frac{80 \cdot 2^{20}}{0.8 \cdot 60 \cdot 10^9} \approx 1.75 \text{ milisegundos.}$$

El módulo generador de acceso realiza un recorrido por las solicitudes de datos de usuario generadas en los módulos de acceso. Cuando un módulo de acceso recibe una solicitud de datos de usuario generada desde el módulo generador de acceso, el módulo de acceso es responsable de realizar solicitudes de fragmentos a los nodos de almacenamiento y recopilar cargas útiles de respuesta; Por lo tanto, debe tener un rendimiento bastante alto. Nuestra implementación de un módulo de acceso utiliza una pila HTTP personalizada, que canaliza las solicitudes y mantiene las conexiones estáticamente vivas durante largos períodos de tiempo. Las solicitudes de fragmentos se cronometran para evitar inundar el conmutador con datos de respuesta y reducir significativamente el riesgo de pérdida de paquetes. La red del servidor también se sintonizó. Estos cambios en conjunto resultaron en una configuración que tiene baja latencia de respuesta y es capaz de saturar completamente los enlaces de red.

Pruebas de rendimiento de acceso

El prototipo se utilizó para comparar el rendimiento de acceso de los sistemas líquidos y los sistemas de código pequeño. El objetivo era evaluar el rendimiento del acceso utilizando implementaciones de sistemas líquidos, es decir, comprender el impacto en el rendimiento de solicitar pequeños fragmentos (o porciones de fragmentos) de una gran cantidad de servidores de almacenamiento. Simplificando, se evaluó el impacto de la

red en el rendimiento del acceso. Los datos de fragmentos generalmente estaban en caché y, por tanto, los tiempos de acceso al disco no eran parte de la evaluación. Tampoco se evaluó el costo computacional de la decodificación.

La Figura 4 muestra la configuración usada para probar la velocidad de acceso. Cada servidor físico tiene un enlace de red dúplex completo de 10 GbE que se conecta a través de un conmutador a todos los otros servidores blade. Las pruebas muestran que se pueden saturar todos los enlaces de 10 Gbps del conmutador simultáneamente. Ocho de los 14 servidores físicos ejecutan accesores, y uno de los ocho también ejecuta el generador de acceso. Los seis servidores restantes ejecutan instancias del software del nodo de almacenamiento. Esta configuración nos permite probar el sistema bajo una carga que satura la capacidad de red a los nodos de almacenamiento: la capacidad de red agregada entre el conmutador y los accesores es de 80 Gbps, que es más que la capacidad de red agregada de 60 Gbps entre el conmutador y los nodos de almacenamiento, por lo que el cuello de botella es de 60 Gbps entre el conmutador y los nodos de almacenamiento.

Durante las pruebas de rendimiento de acceso no hay fallas de nodo, todos los fragmentos n para cada objeto están disponibles y el proceso de reparación está deshabilitado. Para todas las pruebas, se ejecutan 67 instancias de nodo de almacenamiento en cada servidor de almacenamiento, que emula un sistema de 402 nodos. Se operó con una sobrecarga de almacenamiento de 33.3%, se usa un código grande (402, 268, 134) para el sistema líquido y un código pequeño (9, 6, 3) para el sistema de código pequeño. Se probó con tamaños de solicitud de datos de usuario de 10 MB y 100 MB.

Se ejecutó el prototipo de acceso en tres configuraciones diferentes. La configuración "Liq" modela el acceso del sistema líquido a los datos del usuario. Para una solicitud de datos de usuario de tamaño s, el descriptor de acceso solicita porciones de fragmentos $k + E$, cada una de tamaño $s = k$, de un subconjunto aleatorio de nodos de almacenamiento distintos, y mide el tiempo hasta que se reciben las primeras k respuestas completas (cualquier restante a E las respuestas son descartadas silenciosamente por el descriptor de acceso). Cada porción de fragmento tiene un tamaño de alrededor de 25.5 KB cuando $s = 10$ MB, y alrededor de 255 KB cuando s = 100 MB. Usamos $E = 30$ y, por lo tanto, el tamaño total de las porciones de fragmentos solicitadas es de alrededor de 10.75 MB cuando $s = 10$ MB, y alrededor de 107,5 MB cuando $s = 100$ MB.

La configuración "SC" modela el acceso normal de los datos de usuario del sistema de código pequeño, es decir, los datos de usuario solicitados se almacenan en un nodo de almacenamiento que está actualmente disponible. Para una solicitud de datos de usuario de tamaño s, el descriptor de acceso solicita una porción de fragmento de tamaño s de un nodo de almacenamiento seleccionado al azar, y mide el tiempo hasta que se recibe la respuesta completa. El tamaño total de la porción de fragmento solicitada es de 10 MB cuando s = 10 MB y de 100 MB cuando s = 100 MB.

La configuración "SC-Deg" modela el acceso degradado del sistema de código pequeño a los datos del usuario, es decir, los datos del usuario solicitados se almacenan en un nodo de almacenamiento que actualmente no está disponible. Para una solicitud de datos de usuario de tamaño s, el descriptor de acceso solicita k porciones de fragmentos de tamaño s de un subconjunto aleatorio de nodos de almacenamiento distintos y mide el tiempo hasta que se reciben las primeras k respuestas completas. El tamaño total de las porciones de fragmentos solicitadas es de 60 MB cuando s = 10 MB, y 600 MB cuando s = 100 MB. La mayoría de los datos del usuario se almacenan en los nodos de almacenamiento disponibles y, por lo tanto, la mayoría de los accesos son normales, no degradados, en operación de un sistema de código pequeño. Por lo tanto, generamos la carga deseada en el sistema con accesos normales a los datos del usuario, y ejecutamos accesos degradados a una velocidad que agrega solo una carga agregada nominal al sistema, y solo los accesos miden los tiempos para los accesos degradados. Similar a la configuración "Liq", podríamos haber solicitado porciones de fragmentos $k + 1$ para la configuración "SC-Deg" para disminuir la variación en los tiempos de acceso, pero a expensas de una mayor transferencia de datos a través de la red. Las figuras 4 y 5 muestran los resultados del tiempo de acceso obtenidos con las configuraciones anteriores bajo distintas cargas del sistema. El tiempo promedio para los accesos al sistema líquido es menor que para los accesos normales al sistema de código pequeño en la mayoría casos, excepto bajo una carga muy ligera cuando son similares en promedio. Aún mejor, la variación en el tiempo es sustancialmente menor para los sistemas líquidos que para los sistemas de códigos pequeños

bajo todas las cargas. Los accesos al sistema liquido son mucho más rápidos (y consumen menos recursos de red) que los accesos degradados del sistema de código pequeño.

La interpretación de estos resultados es que los sistemas líquidos distribuyen la carga por igual, mientras que los sistemas de código pequeños tienden a estresar los nodos de almacenamiento individuales de manera desigual, lo que lleva a puntos críticos. Con sistemas de código pequeños, las solicitudes a nodos muy cargados resultan en tiempos de respuesta más lentos. Los puntos calientes se producen en dichos sistemas cuando se cargan de manera apreciable, incluso cuando las solicitudes de fragmentos se distribuyen uniformemente al azar. Debería esperarse que, si las solicitudes no son uniformes, pero algunos datos son significativamente más populares que otros, los tiempos de respuesta para sistemas de código pequeños fueran aún más variables.

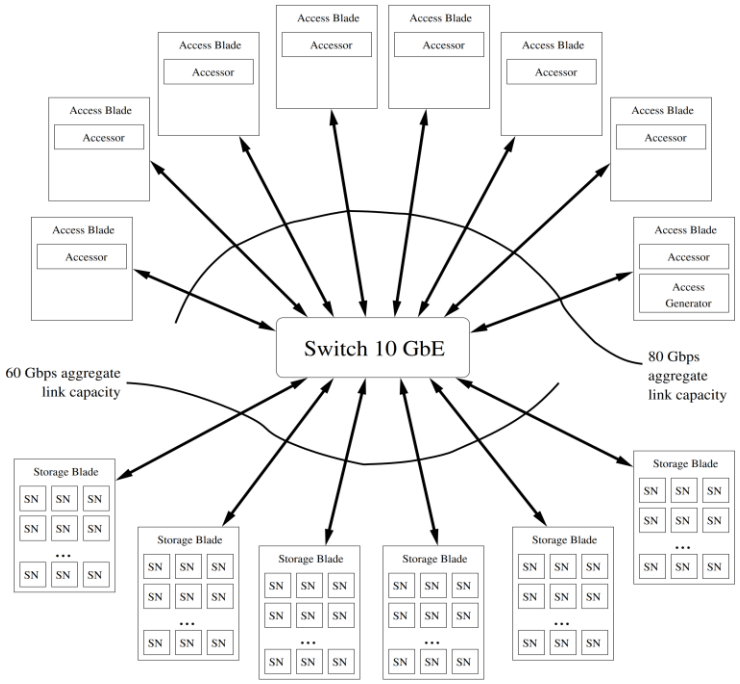

Figura 4. Configuración utilizada para pruebas de rendimiento de acceso

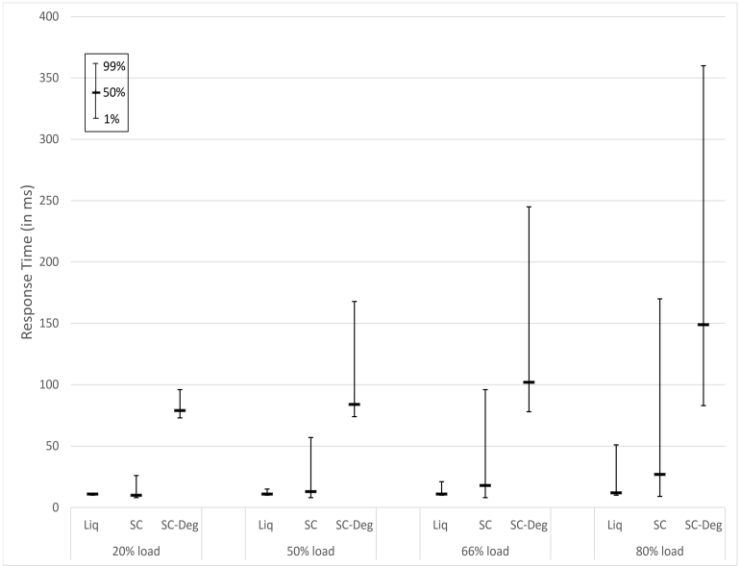

Figura 5. Resultados de acceso para solicitudes de objetos de 10 MB.

Pruebas de reparación del prototipo y verificación del simulador.

Se partió de conjuntos de parámetros realistas, y luego se procedió a acelerarlos en un factor grande (ej., 1 millón) de tal manera se podría ejecutar el proceso de reparación del prototipo y observar la pérdida de objetos en una cantidad de tiempo realista. Se ejecutó el proceso de reparación del prototipo en una configuración de este tipo, donde las fallas de los nodos fueron generadas artificialmente por software. Las estadísticas medidas resultantes, en particular el MTTDL, se compararon con una ejecución equivalente del simulador de reparación del sistema líquido. Esto permitió verificar que las estadísticas del simulador de reparación del sistema líquido coincidían con las del proceso de reparación del prototipo y también con las predicciones analíticas.

X. CONCLUSIONES

Se introdujo un nuevo y exhaustivo enfoque para el almacenamiento distribuido, los sistemas líquidos, que permiten combinaciones flexibles y esencialmente óptimas de confiabilidad de almacenamiento, sobrecarga de almacenamiento, reparación de uso de ancho de banda y rendimiento de acceso. Los ingredientes clave de un sistema líquido son un código grande de baja complejidad, una organización de almacenamiento de flujo y una estrategia de reparación diferida (lenta). El diseño del regulador de reparación que se presentó proporciona una mayor robustez a los sistemas líquidos contra fallos de nodo variables y / o inesperados. Las simulaciones de reparación y acceso establecen que un sistema líquido excede significativamente el rendimiento de los sistemas de códigos pequeños en todas las dimensiones, y permite compensaciones superiores de funcionamiento y funcionamiento en función de los requisitos específicos de implementación de almacenamiento.

Se dirigió a los aspectos prácticos a tener en cuenta al momento de implementar un sistema líquido y proporcionamos una arquitectura de ejemplo. Un sistema líquido elimina los puntos críticos de red y de cómputo y la necesidad de reparación urgente de infraestructura fallida. Si bien no se detalla en el documento, se cree que un sistema líquido proporciona una flexibilidad de distribución geográfica superior y es aplicable a todo tipo de arquitecturas / casos de uso de objetos, archivos, escalado horizontal, hiperconvergentes y HDD / SSD. También hay optimizaciones en las que el equipo ha trabajado que implican la pregeneración de EFI y dan como resultado mejores eficiencias de almacenamiento en los nodos de almacenamiento.

Se realizó un análisis detallado y comparativo entre las diferentes características de los sistemas de código pequeño y los sistemas líquidos al momento de realizar la reparación dejando, de esta manera, en evidencia las ventajas más importantes que presentan estos últimos sistemas y que demuestran el potencial en cuanto a fiabilidad, eficiencia y autonomía en una posible implementación.

XI. REFERENCIAS

[1] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. Fahim ul Haq, M. Ikram ul Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas. 2011. Windows azure storage: A highly available cloud storage service with strong consistency. In Symposium on Operating System Principles.

[2] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. 2012. Erasure coding in 10indows azure storage. In USENIX Annual Technical Conference.

[3] M. Luby. Capacity Bounds for Distribute Storage. submitted to IEEE Transactions on Information Theory, October 2016.

[4] M. Sathiamoorty, M. Asteris, D. Papailiopoulos, A. Dimakis, R. Vadali, S. Chen, and D. Borthakur. XORing Elephants: Novel Erasure Codes for Big Data. Proceedings of the VLDB Endowment, pp. 325-336 Vol. 6, No. 5, 2013.