InGenio Journal
Revista de Ciencias de la Ingeniería de la Universidad Técnica Estatal de Quevedo
https://revistas.uteq.edu.ec/index.php/ingenio
e-ISSN: 2697-3642

Análisis del rendimiento de la paralelización del algoritmo Reed-Solomon

(Analyzing parallelization performance of Reed-Solomon algorithm)

Fabricio R. Marcillo[1,5], Raúl H. Palacios[2], Antonio F. Díaz[1],
Jefferson R. Herrera[3], Ronald D. Camacho[4]
[1] Universidad de Granada, Granada, España
[2] Universidad Autónoma del Estado de Hidalgo, Hidalgo, México
[3] Universidad de las Artes, Guayaquil, Ecuador
[4] Universidad Técnica Estatal de Quevedo, Quevedo, Ecuador
[5] Universidad Técnica de Cotopaxi, La Maná, Ecuador
fmarcillo@correo.ugr.es, raul_palacios@uaeh.edu.mx, afdiaz@ugr.es,
jefferson.herrera@uartes.edu.ec, rcamachor@uteq.edu.ec

Resumen: Los sistemas de almacenamiento distribuido permiten resolver la fuerte demanda de almacenamiento de datos que requiere la sociedad actual. Es por ello que surgen nuevos retos relacionados con la recuperación de datos basada en código de borrado. En este artículo se presenta la paralelización del algoritmo Reed-Solomon a través de hilos. La evaluación se ha realizado en un sistema BLADE, la ejecución del algoritmo se ha realizado en una configuración de 1, 2, 4 y 8 hilos para comprobar el comportamiento del algoritmo. En cuanto a los resultados, se observa que se reducen considerablemente los tiempos requeridos para el procesamiento de los algoritmos tanto para codificación como para decodificación.

Palabras clave: Fiabilidad, tolerancia a los fallos, códigos de control de errores, Reed-Solomon.

Abstract: Distributed storage systems allow to solve the strong demand of data storage required by today's society, this is because new challenges arise related to data recovery based on erasure code. This article presents the parallelization of the Reed-Solomon algorithm through threads. The evaluation was made in a BLADE system, the execution of the algorithm has been done in a configuration of 1, 2, 4 and 8 threads to check the behavior of the algorithm. Regarding the results, it was observed that the times required for processing the algorithms for both encoding and decoding are considerably reduced**.**

Keywords: Reliability, fault tolerance, error control codes, Reed-Solomon.

1. INTRODUCCIÓN

Los sistemas de almacenamiento distribuido dan soporte de disponibilidad de los datos a las aplicaciones de computación de altas prestaciones que además requieren una rápida y eficiente distribución de datos entre los nodos de almacenamiento garantizando resiliencia de estos sistemas. La replicación de datos (*Data Replication*, DR) [1], [2] y la implementación de algoritmos de código de borrado (*Erasure Codes*, EC) [3], [4], [5] garantizan que las aplicaciones dispongan de los datos para la ejecución de sus funciones.

Por un lado, la replicación de datos, aunque podría ser un proceso simple, implica elevado consumo de recursos, en particular los costos de almacenamiento se duplican. Además, hay escenarios en los que dos componentes fallidos (los que tienen ambas copias de un dato) llevan

 Recibido (Received): 2020/10/15
DOI: https://doi.org/10.18779/ingenio.v4i1.365 Aceptado (Accepted): 2020/12/08

a la pérdida de datos [6]. Mientras que, los algoritmos de código de borrado, en caso de fallo en el sistema de almacenamiento, permite la reconstrucción de datos a partir del uso de información almacenada en otro punto de almacenamiento (disco duro o nodo de almacenamiento). Es decir, es un método que protege los datos de tal manera que éstos se dividen en fragmentos, se distribuyen y se codifican con piezas de datos redundantes, y se almacenan en un conjunto de diferentes ubicaciones. En [7] se describe el funcionamiento de los EC. Es decir, en terminología matemática, la protección que ofrecen los EC se podría representar como:

$$n = k + m \tag{1}$$

Donde:

k: cantidad original de datos o símbolos.

m: símbolos redundantes que se agregan para proporcionar protección contra fallas.

n: número total de símbolos creados después del proceso de codificación de borrado.

Tecnológicamente, las capacidades de los discos y las densidades de grabación están creciendo más rápidamente que las velocidades de transferencia de los discos. A pesar de ello, el problema de la disponibilidad se complica cuando tenemos múltiples unidades que pueden fallar en cualquier momento. Utilizar técnicas convencionales de recuperación puede derivar en un serio retraso en la disponibilidad de los servicios de almacenamiento y por ende, fallo en las aplicaciones.

Expuesto lo anterior, surgen retos relacionados con la recuperación basada en código de borrado. Recientemente los códigos de reparación local y los códigos de regeneración son líneas de investigación que permite buscar un equilibrio entre almacenamiento y ancho de banda necesario para la reparación de datos. En este ámbito están trabajando grupos como *Network Coding & Reliable Communications Group* del MIT liderados por Muriel Médard, el grupo de James S. Plank de la Universidad de Tennessee y empresas líderes en almacenamiento como *Netapp* entre otros.

En el presente trabajo se realiza un análisis de rendimiento de la paralelización del algoritmo Reed-Solomon. La paralelización se realiza en función del número de hilos para ejecutar el proceso de codificación y decodificación que se detalla en el siguiente apartado. Según la literatura, los algoritmos de código de borrado son aplicados para almacenamiento distribuido fiable [8], memoria tolerante a fallas [9] y reconstrucción de contenido a partir de datos ampliamente distribuidos [10], [11], [12].

El resto del artículo se estructura de la siguiente manera, en la siguiente sección se analiza el algoritmo Reed-Solomon (configuración y descripción del algoritmo); posteriormente se presenta la sección de evaluación y se describe la arquitectura usada, la configuración para las pruebas; enseguida se presenta la sección de resultados de codificación y decodificación de acuerdo a la configuración para evaluación; finalmente se dan las conclusiones generales de la investigación realizada.

2. ANÁLISIS DEL ALGORITMO REED-SOLOMON

En el presente trabajo se realiza un análisis de rendimiento de la paralelización del algoritmo Reed-Solomon en un escenario donde se realiza la reconstrucción de datos perdidos a partir de datos distribuidos en el almacenamiento de la configuración.

2.1. Configuración de Reed-Solomon

Inicialmente, se utiliza la Ecuación (1) para el proceso de codificación y decodificación, además de la librería *Jerasure* [13]. La propuesta permite recuperación de bloques de un archivo

que previamente se divide en *k* bloques (Figura 1) de tamaño *x*, y en el proceso de codificación se crean *m* bloques de paridad para la recuperación de bloques perdidos o dañados. De tal manera que, se utiliza un espacio de almacenamiento de *storage_space* = (*k* + *m*)

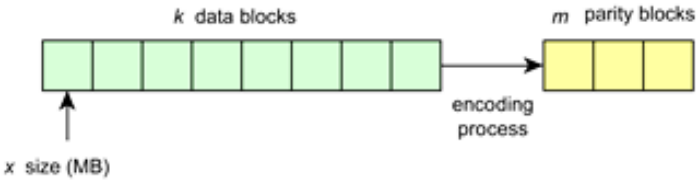

Figura 1. Proceso de codificación *n* = *k* + *m*.

2.2. Descripción de Reed-Solomon

Se ha analizado el rendimiento de una biblioteca práctica de alto rendimiento, Jerasure, que realiza la codificación Reed-Solomon. En cuanto a los códigos de borrado, dos métricas de rendimiento son importantes: la eficiencia del almacenamiento y la tolerancia a fallas. *Erasure Code Reed-Solomon* implica una compensación entre los dos. La eficiencia del almacenamiento es un indicador de almacenamiento adicional requerido para asegurar la resistencia, mientras que la tolerancia a fallas es un indicador de la posibilidad de recuperación en caso de fallas de los elementos, con el fin de aprovechar el máximo rendimiento del hardware se utilizó la multitarea, o la ejecución de múltiples programas y procesos al mismo tiempo, es asistido por el multihilo. Esto permite cambiar rápidamente en el entorno local de tamaños de archivos y optimizar los procesos de codificación y decodificación respectivamente. Además, con el fin de garantizar la integridad de la información restaurada se aplicó el algoritmo SHA-256. En la Figura 2 se muestra la implementación del algoritmo para todos los casos de estudio de este trabajo.

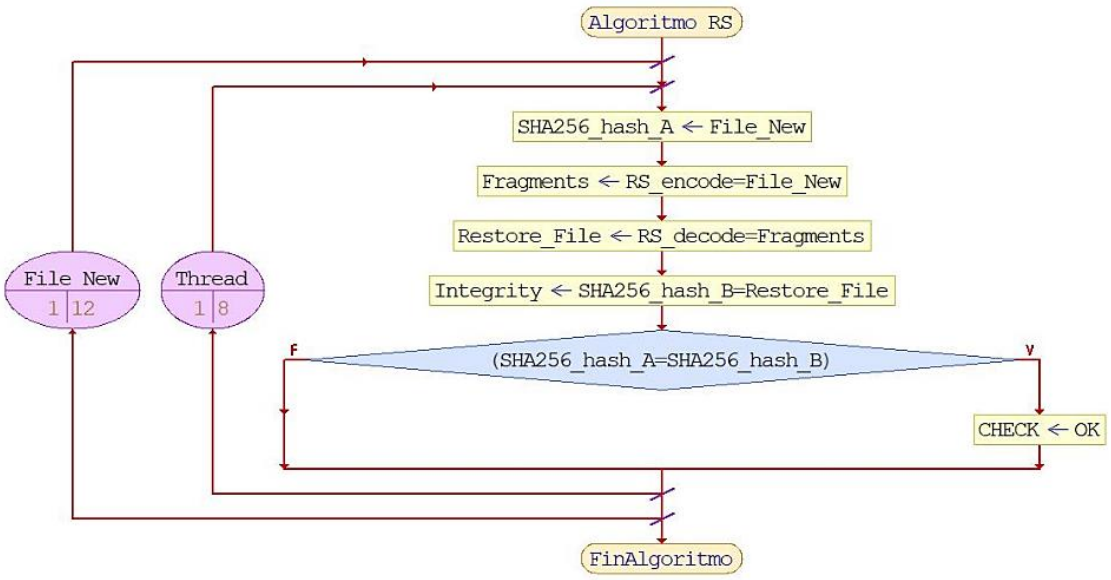

Figura 2. Algorithm Reed-Solomon (jerasure).

3. EVALUACIÓN DEL RENDIMIENTO

En este apartado en primer lugar se describe la configuración hardware empleada para las pruebas. Así mismo, se menciona la configuración de pruebas realizadas en el algoritmo estudiado, donde se dan valores para *k* y *m*.

3.1. Arquitectura local

La evaluación se ha realizado en un Sistema Cisco UCS 5108-AC2 Blade Server Chassis, con procesador Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.2 GHz, memoria RAM de 128GB y unidad de almacenamiento SCSI con capacidad de almacenamiento de 1.2TB, velocidad de rotación de disco duro de 10,000RPM velocidad de transferencia interfaz del disco de 12GB/s.

3.2. Configuración

En primer lugar, se definen los valores para *k* y *m*. En el análisis *k* toma valores de 4, 6 y 10 que se refiere al número de bloques de datos originales. Y el valor de *m* es igual a 2, 3 y 4 respectivamente para cada valor de *k*. el valor de *m* son los símbolos redundantes que se agregan para proporcionar protección contra fallas en caso de que algún valor de *k* se pierda en el almacenamiento.

Así mismo, se realizan pruebas con diferente cantidad de hilos para los procesos de codificación y decodificación. Para este análisis se realizan pruebas con 1, 2, 4 y 8 hilos en cada uno de los procesos de tratamiento de datos.

4. RESULTADOS

A continuación, se muestran los resultados obtenidos para las configuraciones utilizadas en los procesos de codificación y decodificación.

En primer lugar, como se mencionó en la subsección de configuración, en el proceso de codificación se usaron configuraciones en relación al tamaño de bloques de dato originales *k* y del tamaño de bloques de paridad *m* para la recuperación de cualquier bloque perdido. Esta misma relación es usada para realizar la decodificación de datos.

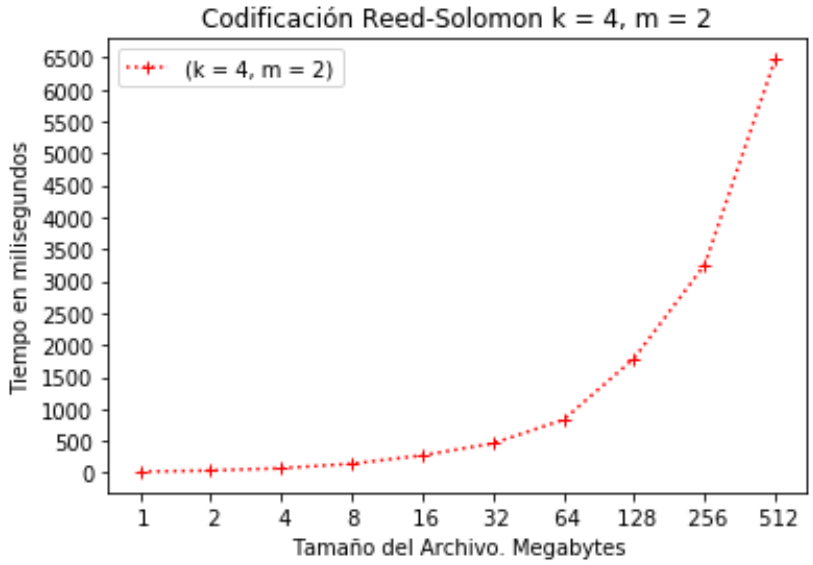

Figura 3 . Configuración RS(4, 2) codificación.

A continuación, se presentan los resultados para ambos procesos. En el eje horizontal se describen los diferentes tamaños de bloques datos y se representa en MB (megabyte), y en el eje vertical el consumo de tiempo para cada tamaño de bloque y se representa en milisegundos.

En la Figura 3 y Figura 4 se muestra el tiempo obtenido en la relación de 4 bloques de datos y 2 bloques de paridad RS(4,2) para recuperación los datos completos en caso de pérdida de 1 o 2 bloques. En el caso de la codificación se observa que el consumo de tiempo es aproximadamente de 6.5 segundos.

En el caso de la Figura 4, la configuración permite una tolerancia de 2 fallos. Esto significa que, si en la configuración existe una pérdida de 3 bloques, el sistema no podrá recuperar los datos completos desde los bloques de paridad. En la gráfica se observa que el sistema consume 3.5 y 6.5 segundo para recuperación de 1 o 2 fallos respectivamente.

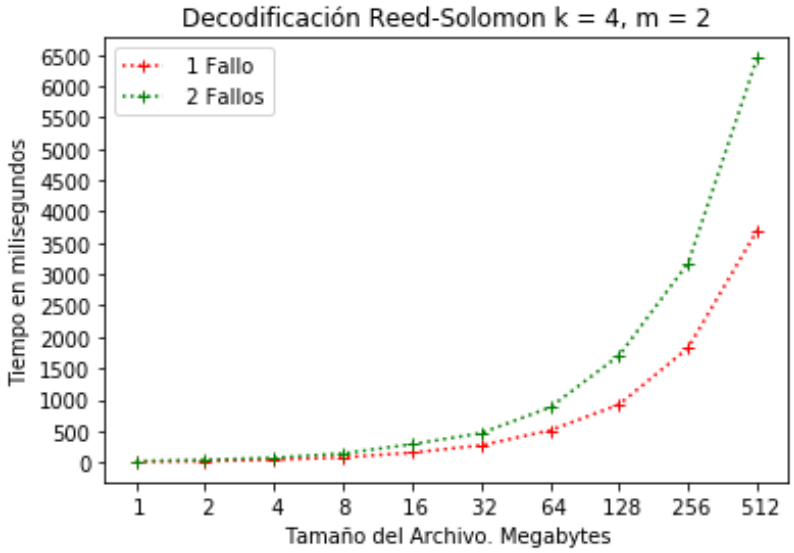

Figura 4. Configuración RS(4, 2) decodificación**.**

La Figura 5 y Figura 6 representa el tiempo consumido para la configuración RS(6, 3). En caso de la codificación se observa que el consumo de tiempo está cerca de los 8.5 segundos con el tamaño máximo de bloque de 512MB.

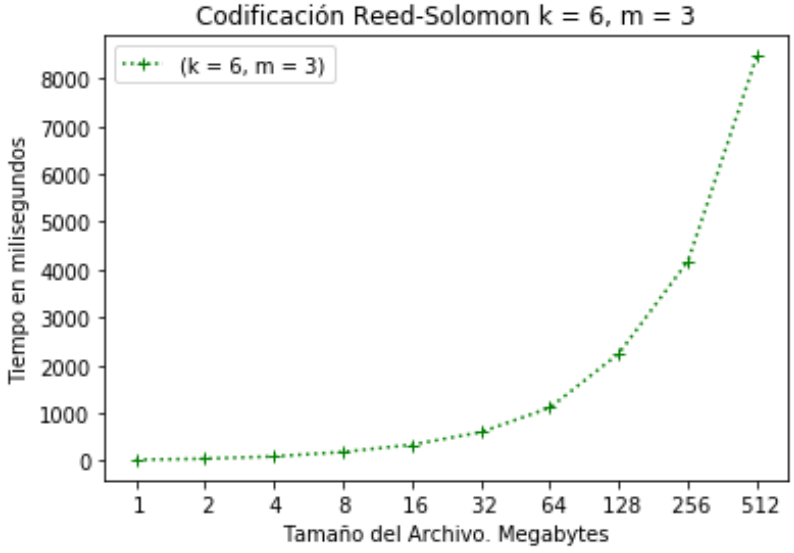

Figura 5. Configuración RS(6, 3) codificación**.**

Así mismo, la configuración permite recuperar de 3 fallos (ver Figura 6) en el sistema. Es decir, es posible la pérdida de 3 bloques de datos para que el sistema sea resiliente. En este caso, el sistema consume 8.5, 6.5 y 3.0 segundos para recuperación de 1, 2 o 3 fallos respectivamente.

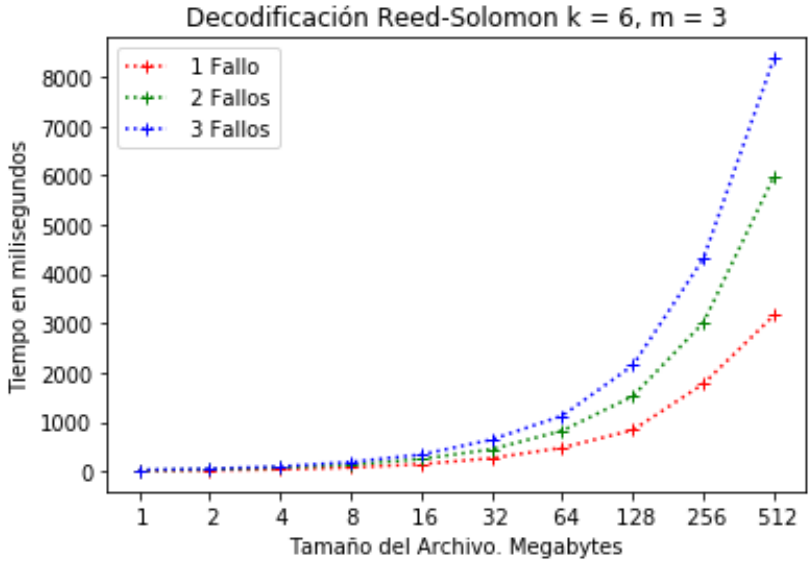

Figura 6. Configuración RS(6, 3) decodificación.

En la Figura 7 y en la Figura 8 se muestran los resultados para codificación y decodificación de la configuración RS(10, 4). En la codificación se consume un tiempo de aproximadamente 11 segundos.

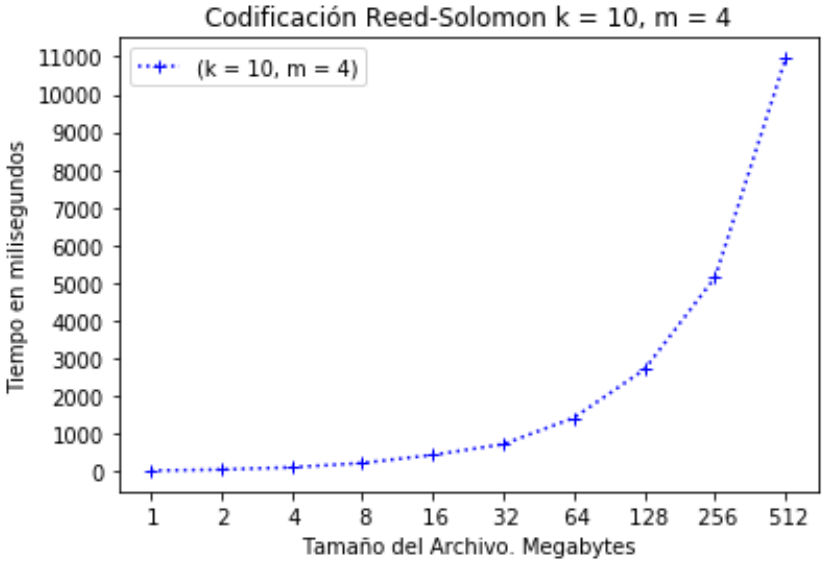

Figura 7. Configuración RS(10, 4) codificación**.**

En la Figura 8 se muestra la decodificación de la configuración RS(10, 4), es decir, el sistema permite hasta 4 fallos. El consumo de tiempo está desde 2.5 hasta 10.5 segundos.

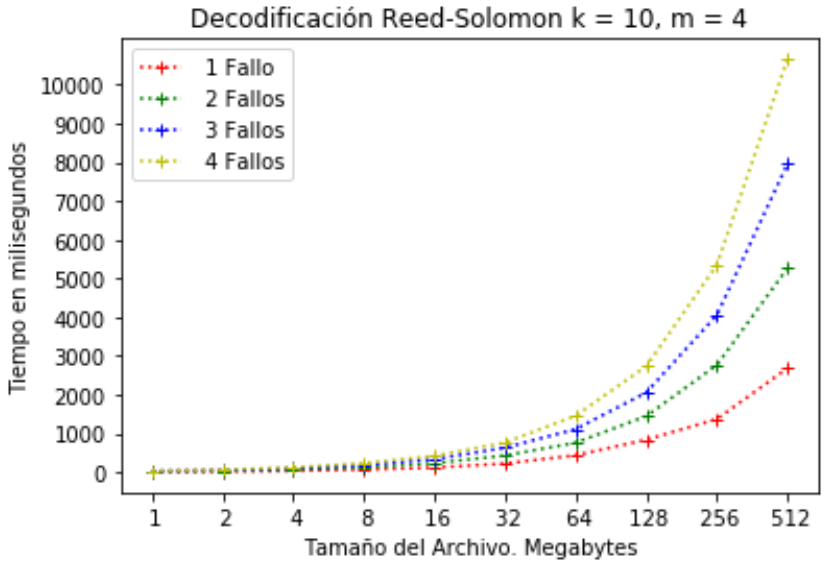

Figura 8. Configuración RS(10, 4) decodificación**.**

El uso de hilos permite distribuir el trabajo en procesos independientes con el fin de equilibrar el uso de recursos del sistema. A continuación, se presentan los resultados con la configuración antes mencionada y con la implementación de hilos que van de 1 a 8 hilos para cada configuración. La Figura 9 y Figura 10 muestran los resultados para la configuración RS(4, 2) con hilos. En esta configuración se observa la disminución considerable de tiempo cuando se usan 8 hilos para el tamaño de bloque máximo (512MB). A diferencia de la Figura 3, en la Figura 9 el consumo de tiempo máximo es de menos de 1 segundo.

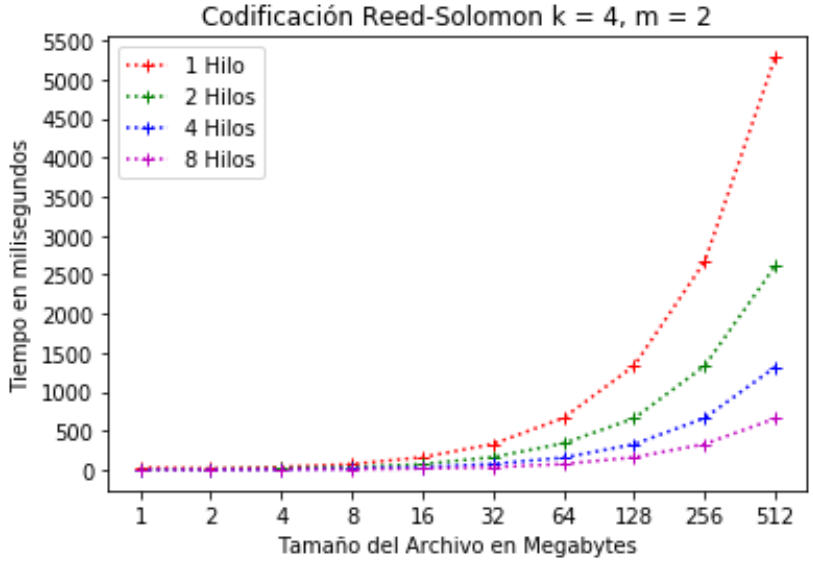

Figura 9. Configuración RS(4, 2) codificación con hilos.

En cuanto a la decodificación, en la Figura 10 se presenta el consumo de tiempo para 1 y 2 fallos. Se observa que para 2 fallos y 8 hilos, el consumo de tiempo es de ligeramente arriba de 1 segundo.

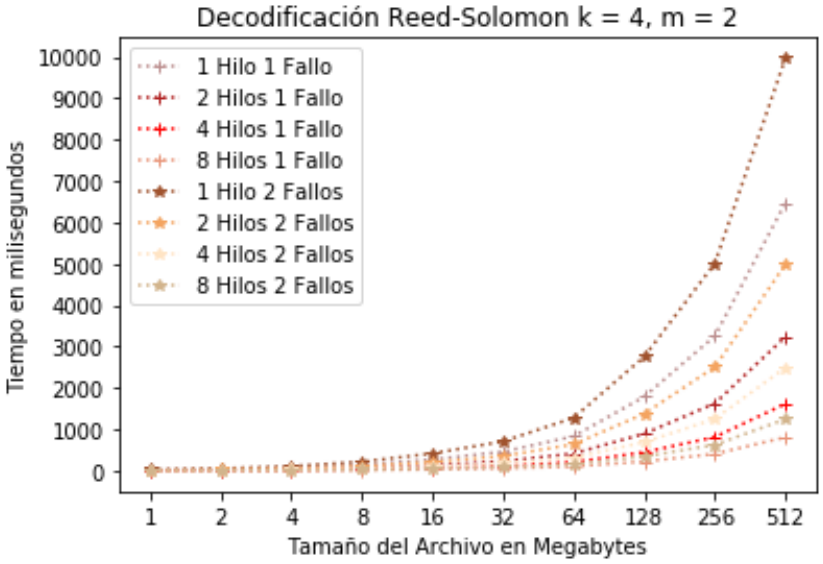

Figura 10. Configuración RS(4, 2) decodificación con hilos**.**

En el caso de la Figura 11, la configuración crea 6 bloques de datos originales y 3 bloques de paridad. El consumo de tiempo alcanza ligeramente 1 segundo cuando se usa la configuración de 8 hilos.

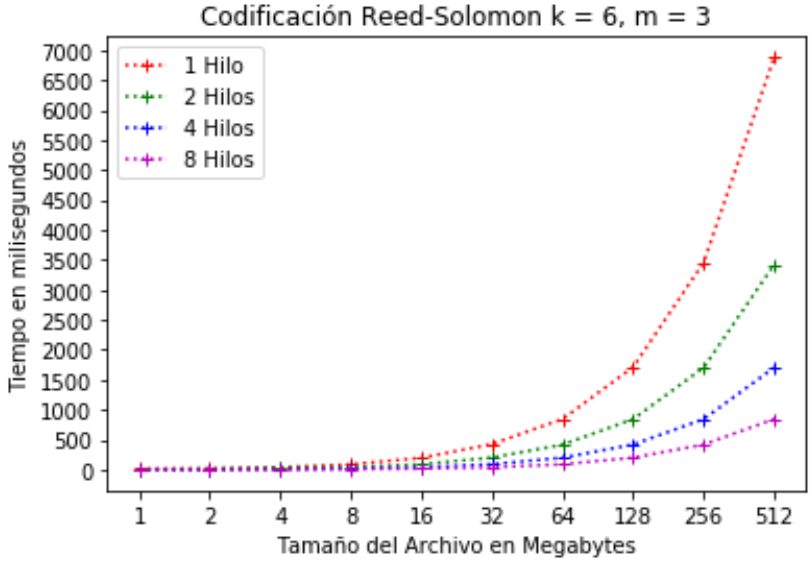

Figura 11. Configuración RS(6, 3) codificación con hilos**.**

En la Figura 12 se muestra el tiempo para el número de hilos y de fallos permitidos en la configuración. Se observa que, para 8 hilos y 1 fallo el consumo es de poco menos de 1.5 segundos, 8 hilos y 2 fallos está encima de los 1.5 segundos y, 8 hilos y 3 fallos es de unos 3 segundos.

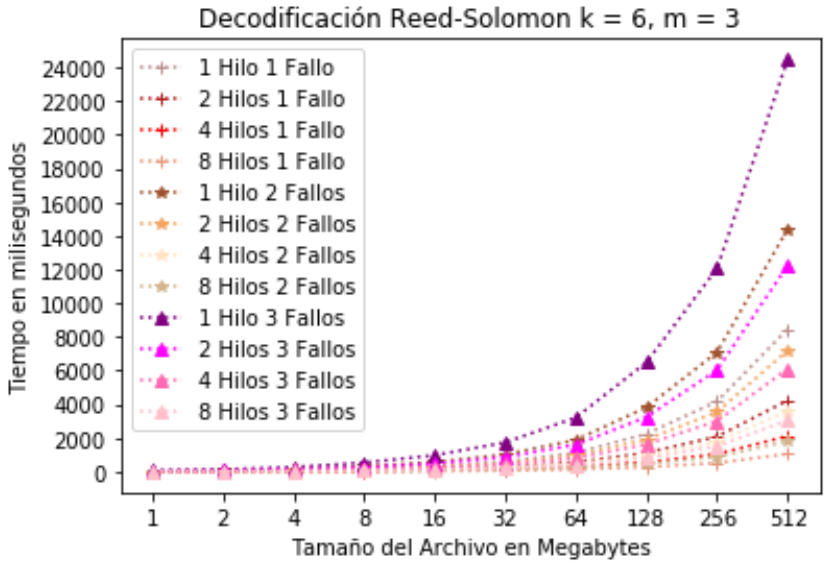

Figura 12. Configuración RS(6, 3) decodificación con hilos.

En la Figura 13 se presentan los resultados de la configuración RS(10, 4). Es decir, el archivo principal de fragmenta en 10 bloques de datos originales y se agregan 4 bloques de paridad. En este caso, se observa que en la configuración con 8 hilos se consume alrededor de 1 segundo en la codificación.

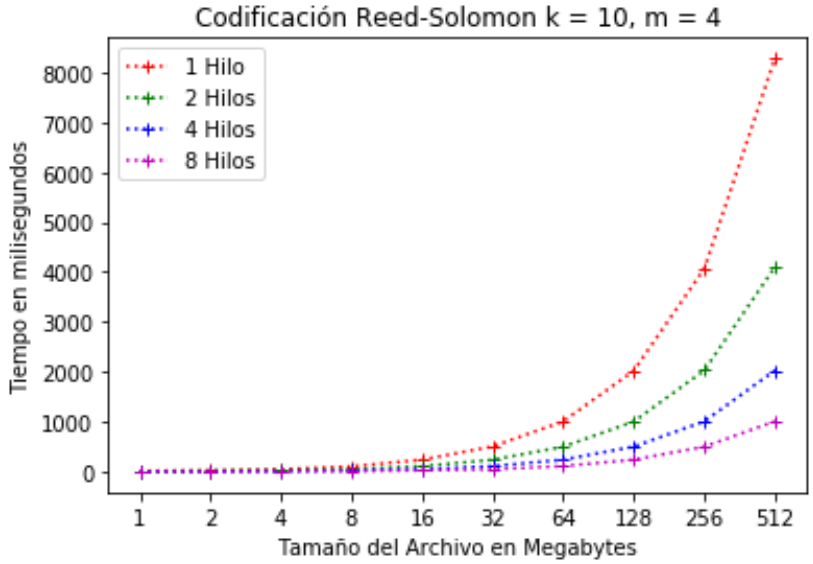

Figura 13. Configuración RS(10, 4) codificación con hilos**.**

En la Figura 14 se observa los resultados con diferente tolerancia de fallo de la configuración RS(10, 4). Se muestra que con 8 hilos y 1 fallos se consume unos 2 segundos, 8 hilos y 2 fallos unos 3 segundos, 8 hilos y 3 fallos consumen poco más de 3 segundos y 8 hilos y 4 fallos consume unos 6 segundos. Estos casos son para tamaño de bloque de 512MB.

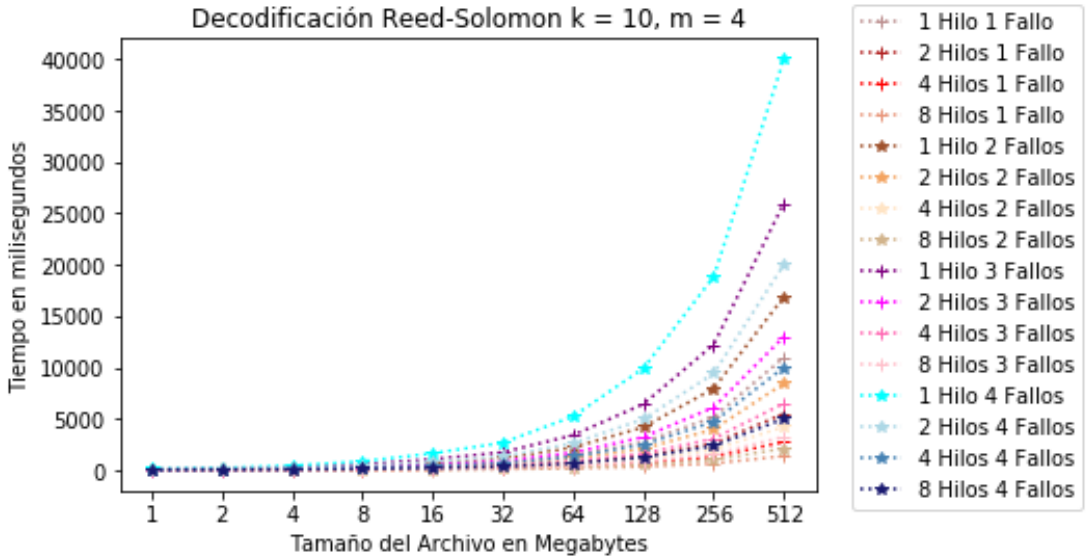

Figura 14. Configuración RS(10, 4) decodificación con hilos**.**

En el caso de uso de hilos, se observa que, en la codificación el consume de tiempo está en 1 segundo para el tamaño de bloque de 512MB. Caso contrario en la decodificación, donde el tiempo difiere debido a que para cada configuración se crean 2, 3, o 4 bloques de paridad para recuperar los datos. Por lo tanto, el sistema consume más recursos para hacer la reparación de todos los bloques originales de datos desde los bloques de paridad.

5. CONCLUSIONES

Los hilos se distinguen de los tradicionales procesos, donde, los procesos son generalmente independientes, llevan mucha información de estados, e interactúan sólo a través de mecanismos de comunicación dados por el sistema. Por otra parte, muchos hilos generalmente aprovechan mejor los recursos del procesador.

En muchos de los sistemas Cisco UCS 5108-AC2 Blade Server Chassis que proveen facilidades para los hilos, es más rápido cambiar de un hilo a otro dentro del mismo proceso, que cambiar de un proceso a otro. En este contexto en el presente trabajo se enmarca en la optimización del algoritmo *Erasure Code Reed-Solomon* utilizando diferentes parametrizaciones en el paralelismo con énfasis en la eficiencia del almacenamiento y la tolerancia a fallas; obteniendo resultados óptimos en la ejecución de la codificación y decodificación.

El uso de paralelización mediante hilos reduce considerablemente los tiempos de ejecución de los algoritmos de codificación y decodificación Reed-Solomon para distintos tamaños de procesamiento.

AGRADECIMIENTOS: Esta investigación fue financiada por el Ministerio de Ciencia, Innovación y Universidades de España (MICINN), subvención número PGC2018-096663-B-C44 junto con el Fondo Europeo de Desarrollo Regional (FEDER) y, por el Programa para el

Desarrollo Profesional Docente, para el Tipo Superior (PRODEP) de México con la financiación PROMEP/103.5/13/6475 UAEH-146.

REFERENCIAS

[1] Palacios, R. H., Díaz, A. F., Anguita, M., Ortega, J., & Rodríguez-Quintana, C. (2017). High-throughput multi-multicast transfers in data center networks. The Journal of Supercomputing, 73(1), 152-163.

[2] Palacios, R. H., Rodríguez-Quintana, C., Díaz, A. F., Anguita, M., & Ortega, J. (2017). Evaluation of redundant data storage in clusters based on multi-multicast and local storage. The Journal of Supercomputing, 73(1), 576-590.

[3] Aguilera, M. K., Janakiraman, R., & Xu, L. (2005, June). Using erasure codes efficiently for storage in a distributed system. In 2005 International Conference on Dependable Systems and Networks (DSN'05) (pp. 336-345). IEEE.

[4] Hafner, J. L. (2005, December). WEAVER Codes: Highly Fault Tolerant Erasure Codes for Storage Systems. In Fast (Vol. 5, pp. 16-16).

[5] Khan, O., Burns, R. C., Plank, J. S., Pierce, W., & Huang, C. (2012, February). Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads. In FAST (p. 20).

[6] Plank, J. S. (2013). Erasure codes for storage systems: A brief primer. ; login:: the magazine of USENIX & SAGE, 38(6), 44-50.

[7] Weatherspoon, H., & Kubiatowicz, J. D. (2002, March). Erasure coding vs. replication: A quantitative comparison. In International Workshop on Peer-to-Peer Systems (pp. 328-337). Springer, Berlin, Heidelberg.

[8] Rashmi, K. V., Shah, N. B., Ramchandran, K., & Kumar, P. V. (2017). Information-theoretically secure erasure codes for distributed storage. IEEE Transactions on Information Theory, 64(3), 1621-1646.

[9] Yiu, M. M., Chan, H. H., & Lee, P. P. (2017, May). Erasure coding for small objects in in-memory kv storage. In Proceedings of the 10th ACM International Systems and Storage Conference (pp. 1-12).

[10] Sobe, P. (2010, May). Parallel reed/solomon coding on multicore processors. In 2010 International Workshop on Storage Network Architecture and Parallel I/Os (pp. 71-80). IEEE.

[11] Chen, R., & Xu, L. (2020). Practical performance evaluation of space optimal erasure codes for high-speed data storage systems. SN Computer Science, 1(1), 1-14.

[12] Jin, H., Wu, C., Xie, X., Li, J., Guo, M., Lin, H., & Zhang, J. (2019, August). Approximate code: a cost-effective erasure coding framework for tiered video storage in cloud systems. In Proceedings of the 48th International Conference on Parallel Processing (pp. 1-10).

[13] Plank, J. S., Simmerman, S., & Schuman, C. D. (2007). Jerasure: A library in C/C++ facilitating erasure coding for storage applications. Technical Report CS-07-603, University of Tennessee.