Towards an Efficient Storage Workload Modeling with Time-Series Clustering Parallelization

(Reporte de Proyecto)

Edwin F. Boza , Gladys E. Carrillo

10 de abril de 2017

1. Introducción

La evaluación de Sistemas de Almacenamiento Distribuidos, con el objetivo de comparar productos o evaluar mejoras a los mismos, requiere el uso de cargas de trabajo modeladas a partir de sistemas reales [1].

Para realizar el modelamiento, se requiere como insumo una traza de cargas de trabajo, que es una colección de registros que indican los accesos y operaciones que se han efectuado sobre un sistema en producción. Dicha traza, generalmente se la obtiene mediante mecanismos de monitoreo de un sistema de almacenamiento en producción.

El proceso de modelamiento consiste en abstraer características esenciales de objetos en la traza disponible; dichas características deben permitir la reproducción sintética de la carga de trabajo descrita por la mencionada traza.

En el caso de los sistemas de almacenamiento, las trazas de cargas de trabajo contienen la secuencia de accesos a los objetos almacenados, durante un periodo determinado de tiempo. El modelamiento consiste en identificar los patrones de acceso a cada objeto y obtener un modelo que permita generar dicho patrón [2]. Este nuevo modelo, será utilizado para la generación automática de secuencias de acceso que pueden ser utilizadas al momento de la evaluación.

Sin embargo, las trazas de sistemas de almacenamiento con frecuencia contienen accesos a millones de objetos. Mantener y procesar un modelo por cada uno de los objetos requiere una excesiva cantidad de recursos (memoria y procesamiento) y puede afectar el funcionamiento de la herramienta de evaluación. Para abordar este problema, se puede utilizar algoritmos de aprendizaje automático, para agrupar los objetos similares en un número reducido de clases. Finalmente, se procede a obtener un modelo por cada clase de objetos.

Existen varios enfoques para realizar el agrupamiento de objetos similares, basados en sus patrones de acceso. Por ejemplo, en un trabajo previo [3], se utilizó un conjunto de medidas de resumen estadístico como descriptores de las secuencias de interarribos de cada objeto (Fig. 1). Con este conjunto de medidas, se realizó un proceso de clasificación utilizando el algoritmo de K-means.

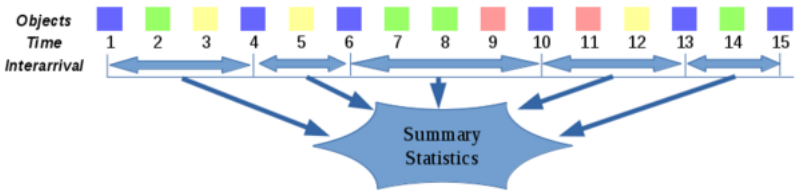

Figura 1: Luego de obtener los tiempos de interarribo para cada objeto, se calcula un conjunto de medidas de resumen estadístico, que son utilizadas para alimentar un algoritmo de clasificación (K-means).

Una nueva propuesta consiste en considerar los tiempos de interarribo de cada objeto, como una serie de tiempo y encontrar grupos o clusters que sean significativos y útiles. Sin embargo, aunque existen varias técnicas y herramientas para agrupar series de tiempo como por ejemplo, el paquete TSclust, del lenguaje de programación R, su rendimiento se ve afectado en gran manera con grandes datasets.

En este trabajo, proponemos el uso de Apache Spark [4] para acelerar el proceso de agrupamiento de las series de tiempo, paralelizando la construcción de la matriz de distancia correspondiente. Adicionalmente exploramos el uso del método de muestra probabilística [5] para reducir el espacio y tiempo requeridos para el proceso de agrupamiento. Mostramos que el uso de un método de agrupación basado en una sencilla paralelización de las series de tiempo conduce a una reducción considerable del tiempo requerido, sin embargo, todavía no es adecuado para ser usado con traces con millones de objetos. Por lo tanto, mostramos que el uso del método de muestreo probabilístico, en combinación con la ya mencionada paralelización, permite realizar la tarea de agrupamiento en tiempo asequible, manteniendo una precisión de hasta 65 %.

2. Trabajos relacionados

Las plataformas para el procesamiento de datos en paralelo, especialmente para el cálculo de matrices de similaridad entre objetos, se emplean en diferentes áreas.

En el campo de la biología, los datos moleculares de alto rendimiento hacen complejas las tareas de agrupamiento debido al mal desempeño del cálculo de la matriz de correlación. Ante este problema, Wang et al. [6] han propuesto un nuevo enfoque para calcular la correlación aplicando un algoritmo efectivo basado en MapReduce para optimizar el almacenamiento y el cálculo de correlación. La propuesta se implementó con RHIPE, un paquete basado en R de Hadoop MapReduce, que transforma las funciones R en trabajos MapReduce. El algoritmo divide los vectores de entrada, que contienen valores de intensidad para un solo sujeto, en bloques de datos que se envían a diferentes Mappers. A continuación, cada bloque de datos se copia en un reductor que calcula los valores de correlación. Como resultado, este algoritmo funciona 30 veces más rápido que la implementación normal de R.

En el ámbito marítimo, Liu [7] propuso una versión paralela de un algoritmo de agrupación para la extracción de patrones de tráfico marítimo a partir de datos del "Sistema de Identificación Automática de Tráfico Marítimo". El algoritmo paralelo fue diseñado e implementado con Apache Spark. El algoritmo paralelo fue diseñado para convertir el problema a uno basado en grafos para aplicar también el API GraphX. Aunque el algoritmo propuesto no tiene un resultado de rendimiento satisfactorio, sirve como punto de partida para buscar otros métodos que permitan reducir la complejidad del algoritmo.

Otros enfoques para el modelado de cargas de trabajo para sistemas de almacenamiento proponen el uso de métodos probabilísticos para reducir los requerimientos de espacio y tiempo necesarios para obtener el modelo.

Wires et al. proponen una nueva estructura de datos, el "counter stack", para aproximar Miss Ratio Curves (MRCs) para producir modelos de carga de trabajo con menores gastos computacionales y de memoria. Los resultados de sus experimentos con grandes cargas de trabajo muestran una reducción del 70 % en el uso de memoria y almacenamiento y los MRCs resultantes con una tasa de error entre 0.002 y 0.02.

Waldspurger y otros [8] introducen SHARDS, un algoritmo para calcular aproximaciones de Miss Ratio Curves, utilizando una técnica de muestreo espacial basada en hash. La evaluación del algoritmo mostró que los MRC pueden construirse en línea usando menos de 10 MB de memoria por carga de trabajo. Además, el error entre las MRC aproximadas y exactas es muy bajo, entre 0.007 y 0.02.

3. Metodología

Nuestra propuesta está dividida en cuatro pasos principales, las cuales se describen en el Algoritmo 1:

Algorithm 1 Pasos para el agrupamiento de series de tiempo

1: Extraer los diez mil objetos más accesados
2: Obtener una muestra representativa de los objetos más accesados.
3: Contruir la matriz de distancia para las series de tiempo correspondientes a los tiempos de interarribos de los objetos muestreados.
4: Ejecutar el proceso de agrupamiento con la matriz de distancia obtenida en el paso 3.

Comenzamos por extraer el subconjunto de los objetos más accesados, ya que representarán la mayor parte de la carga de trabajo. Además, como se muestra en la sección de resultados, la precisión disminuye cuando el agrupamiento se realiza a series de tiempo de tamaños muy diferentes.

En el paso siguiente, se debe obtener una muestra representativa seleccionada aleatoriamente de los objetos más accesibles para reducir los recursos requeridos, mientras se mantiene un alto nivel de precisión en el proceso de agrupación.

Para la tercera fase, la matriz de distancia esperada es una matriz simétrica que contiene la medida de la disimilitud de las series de tiempo entre cada par de objetos. El cálculo de la matriz de distancia es un proceso "perfectamente paralelo", por lo tanto sólo es necesario calcular la distancia entre pares únicos de objetos. Para acelerar este proceso, propusimos un algoritmo paralelo para ser usado con Apache Spark que se detalla a continuación.

Algorithm 2 Cálculo en paralelo de la matriz de distancia

```
1: procedure CalcularMatrizDistancia
2:     Leer lista de objetos como RDD
3:     Combinar objetos para obtener una lista de pares como RDD.
4:     Calcular distancia entre las series de tiempo de cada par de objetos en paralelo.
5:     Completar matriz de distancia con los valores calculados.
6: end procedure
```

En el último paso, para el proceso de agrupación, se propone el uso del algoritmo de agrupamiento K-medoids [9]. Este algoritmo es menos sensible al ruido y valores atípicos que los K-means más comúnmente utilizados; y para el caso de las series de tiempo, K-medoids es más adecuado porque siempre selecciona objetos de la muestra como centroides para los clústers, en lugar de *calcular / crear* un nuevo centroide, que no es una acción confiable con las series de tiempo.

4. Experimentos y Resultados

El conjunto de datos utilizado en los experimentos consiste en un trace de accesos a un sistema de almacenamiento (HDFS) de la empresa Yahoo Inc. El trace contiene 54,600,645 accesos a 4,988,059 objetos únicos. Se realizó un paso de preprocesamiento para obtener los tiempos de interarribos para cada objeto.

Se utilizaron dos servidores para probar la versión paralela del cálculo de la matriz de distancia. La Tabla 1 muestra las características de los servidores. El servidor 1 tiene tres veces más núcleos de CPU que el servidor 2; sin embargo, el último tiene tres veces más RAM que el anterior.

Cuadro 1: Características de los servidores usados en el experimento

	RAM	**Modelo CPU**	**Número de núcleos**
Servidor 1	32GB	Intel Xeon 2.30GHz	72
Servidor 2	96GB	Intel Xeon 2.50GHz	24

Para continuar con el uso de la librería TSclust de R, que se había empleado inicialmente para el cálculo de la matriz de distancia de manera tradicional, se implementó una versión paralela del cálculo de la matriz de distancia usando SparkR(R on Spark). El paquete TSclust, si bien es útil puesto que implementa una variedad de medidas de disimilitud y algoritmos de agrupamiento para series de tiempo, no pudo ser utilizado en la versión paralela del cálculo de la matriz. El código desarrollado [1] no producía ninguna salida después de varios minutos de ejecución, presumiblemente porque la implementación del algoritmo de distancia (Frechet) no podía ser ejecutado en Spark.

Ante este inconveniente se procedió a implementar la matriz de distancia en Spark 2.1.0 utilizando la API de Python versión 2.7.

Como medida de distancia se utilizó inicialmente una implementación en python de la distancia de Frechet [2], sin embargo para series de tiempo de mayor tamaño producía que la ejecución del cálculo de la matriz colapsara. La distancia de Frechet emplea método recursivo utilizando

[1] https://github.com/gladyscarrillo/mcc-adm-bozacarrillo/blob/master/CalculoDistanciaTClustSpark.R
[2] https://www.snip2code.com/Snippet/76076/Fr-chet-Distance-in-Python

valores previamente calculados y también almacenados en una matriz. Ante el inconveniente con el algoritmo de Frechet se buscó otra alternativa de medida de disimilitud, siendo seleccionada una implementación de Python de FastDTW [10]. Esta versión mejorada de Dynamic Time Warping (DTW) [11] permite medir la similitud entre dos secuencias como las series de tiempo con un tiempo O(N). El proceso de agrupamiento se realizó mediante la implementación en Python de K-medoids implementado por Bauckhage [9].

Para iniciar con los experimentos, se desarrolló en Python la versión tradicional del cálculo de la matriz de distancia, esto es mediante el uso de bucles para calcular el valor de cada celda de la matriz.

A continuación se implementó el algoritmo paralelo mostrado en el Algoritmo 2. La implementación en PySpark se muestra a continuación:

```
objetos = sc.textFile(fileName,24)
pairs_data = objetos.cartesian(objetos).filter(lambda (a, b): a <= b)
pairs_data2 = sc.parallelize(pairs_data.collect(),24)
matriz_distancia = pairs_data2.map(calcular_distancia).collect()
df_matriz =pd.DataFrame(matriz_distancia, columns=['obj1','obj2','distance'])
```

Figura 2: Implementación en PySpark del Algoritmo paralelo para calcular matriz de distancia

El objetivo de nuestro primer experimento fue mostrar la mejora en el tiempo obtenida a partir de la paralelización del proceso de cálculo de la matriz de distancia. La tabla 2 muestra los tiempos requeridos para calcular la matriz de distancia para un conjunto aleatorio de 1.000 objetos, usando la distancia FastDTW. El servidor 1, que tiene más núcleos de CPU, obtiene los mejores resultados tanto para métodos secuenciales como paralelos. También muestra un *speedup* de 24,45, que es mayor que el speedup alcanzado por el Servidor 2 con el método paralelo.

Cuadro 2: Tiempo de ejecución para el cálculo de la matriz de distancia para 1000 objetos

	Secuencial	**Spark**	**Speedup**
Servidor 1	28.82 horas	1.05 horas	24.45
Servidor 2	64.91 horas	4.18 horas	15.52

Aunque el servidor 1 tiene tres veces más núcleos de CPU, la velocidad alcanzada por este servidor es aproximadamente 1,77 veces la velocidad obtenida para el servidor 2. Esto puede explicarse por la mayor cantidad de memoria disponible en el segundo servidor. Sin embargo, está claro que los núcleos de la CPU serán el factor principal en la aceleración.

Consideramos nuestro método de paralelización *naïve*, porque sólo nos hemos centrado en la optimización del cálculo de la matriz, que es un problema "perfectamente paralelo". Sin embargo, las funciones para calcular las distancias o las medidas de similitud para las series de tiempo no son paralelas y tienen mayores complejidades de tiempo big O.

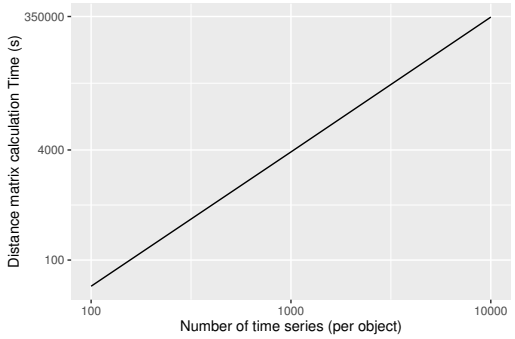

Figura 3: Tiempo de ejecución para calcular la matriz de distancia usando Spark

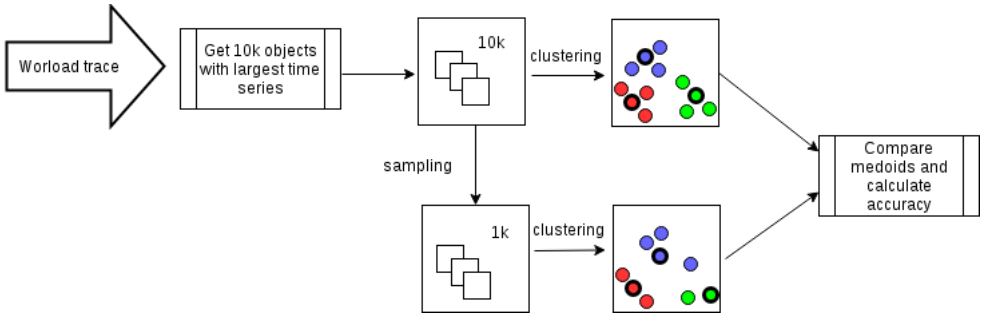

Figura 4: Diseño experimental para la prueba de precisión del modelo propuesto

La figura 3 muestra la evolución del tiempo de ejecución para el cálculo paralelo de la matriz de distancia, con un número creciente de series de tiempo. Las escalas de registros indican un crecimiento exponencial en el tiempo de ejecución observado, a medida que aumenta el número de series de tiempo. El tiempo de ejecución de una muestra de 100 objetos fue 41,64 segundos, para 1.000 objetos fue 3.743,95 segundos y para 10.000 objetos fue 343.140 segundos.

Este comportamiento compromete el uso de este método sencillo de paralelización como el único método para acelerar el agrupamiento de series de tiempo y apoya nuestra decisión de explorar otras técnicas para completar el agrupamiento en un tiempo asequible.

Para evaluar la precisión de nuestro modelo propuesto, diseñamos y realizamos el experimento mostrado en la figura 4.

En el primer paso, se selecciona un subconjunto de 10,000 objetos, correspondiente a los objetos más accedidos. Los objetos más populares, que corresponden a las series de tiempo más grandes, se seleccionan porque la precisión de la agrupación disminuye cuando el rango de tamaños de las series de tiempo está demasiado extendido, como se muestra en la Figura 5

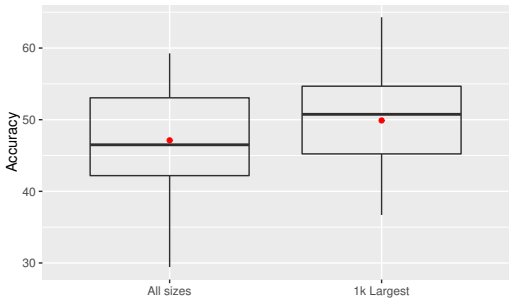

Figura 5: Comparación de la precisión de agrupamiento para objetos de 1000 series de tiempo de varios tamaños comparadas con las series de tiempo más grandes. La precisión aumenta cuando los tamaños de series de tiempo son similares.

Este nuevo conjunto de datos se utilizó para obtener la matriz de distancia basada en el diseño propuesto. Utilizando la matriz de distancias, se realizó un proceso de agrupamiento con algoritmo K-medoides para obtener 10 grupos.

En el paso final, la precisión se mide comparando la eficiencia de los medoides obtenidos en el agrupamiento del subconjunto muestreado, para reproducir el mismo agrupamiento obtenido en el agrupamiento de las series de tiempo de los 10.000 objetos. La figura 6, muestra los resultados de precisión obtenidos con nuestro método experimental; la precisión mediana es 57,1 %, y la precisión máxima es 65,34 %.

5. Conclusiones

En este trabajo, hemos propuesto un enfoque combinado para acelerar el proceso de agrupación de series de tiempo.

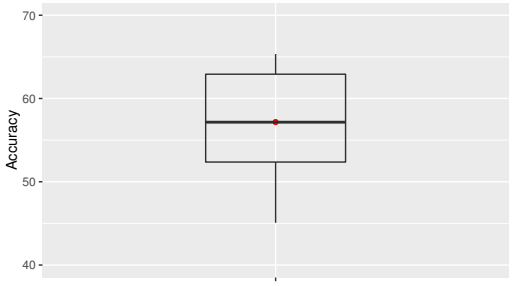

Figura 6: Precisión del Agrupamiento: Mediana = 57.1 %; Max = 65.34 %

Hemos demostrado que el uso del framework Apache Spark [4], para paralelizar la construcción de una matriz de distancia, conduce a una aceleración tan grande como 24, en comparación con el correspondiente algoritmo secuencial.

Además, nuestros experimentos demuestran que es viable usar un método de muestreo probabilístico para reducir los requerimientos de espacio y tiempo para el proceso de agrupamiento, manteniendo una precisión de hasta 65 % .

Referencias

[1] A. Traeger, E. Zadok, N. Joukov, and C. Wright, "A nine year study of file system and storage benchmarking," *ACM Transactions on Storage (TOS)*, vol. 4, no. 2, 2008.

[2] P. P. Ware, T. W. Page Jr, and B. L. Nelson, "Automatic modeling of file system workloads using two-level arrival processes," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 8, no. 3, pp. 305–330, 1998.

[3] C. Abad, M. Yuan, C. Cai, Y. Lu, N. Roberts, and R. Campbell, "Generating request streams on Big Data using clustered renewal processes," *Performance Evaluation*, vol. 70, no. 10, 2013.

[4] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets.," *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[5] B. Kitchenham and S. L. Pfleeger, "Principles of survey research: part 5: populations and samples," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 5, pp. 17–20, 2002.

[6] S. Wang, I. Pandis, D. Johnson, I. Emam, F. Guitton, A. Oehmichen, and Y. Guo, "Optimising parallel r correlation matrix calculations on gene expression data using mapreduce," 2014.

[7] B. Liu, "Parallel maritime traffic clustering based on apache spark,"

[8] C. A. Waldspurger, N. Park, A. T. Garthwaite, and I. Ahmad, "Efficient mrc construction with shards.," 2015.

[9] C. Bauckhage, "Numpy/scipy recipes for data science: k-medoids clustering," *researchgate. net, Feb*, 2015.

[10] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[11] M. Müller, "Dynamic time warping," *Information retrieval for music and motion*, pp. 69–84, 2007.