Comparative study of the estimation of the progress score of MapReduce tasks in speculative schedulers

Apaza Veliz, Ronald Darwin[1], Mamani Aliaga, Alvaro Henry[1]
[1]Universidad Nacional de San Agustín, Arequipa, ronald.dev1@gmail.com,amamaniali@unsa.edu.pe

Abstract– ***Large amounts of data are being generated constantly, for this reason it is necessary to use parallel and distributed systems, however these systems require careful control in the management of their resources, which is why the MapReduce programming model emerges, which using its schedulers is responsible for automatically managing distributed system resources, for example if one cluster node stops working, MapReduce automatically executes the interrupted task on another computer, another similar mechanism is the speculative scheduling, which is responsible for detecting those tasks that have an execution time abnormally prologando and executes copies of these in a computer different from the one that is executing the original task, this is done with the hope that the backup task finish before the original task and in this way reduce the execution time of the entire application, however the current speculative schedulers have some limitations to correctly calculate the progress of the tasks. The calculation of the progress of the tasks is fundamental to determine when a task is straggler and a wrong estimation of the progress could cause that resources of the system are wasted and to prolong the time of execution of all the application, for this reason in this work a comparative to measure the accuracy in estimating the progress of the tasks of various speculative schedulers and finally points out some guidelines that could serve as a guide for future proposals and that through these guidelines achieve a more efficient estimate of progress.***
Keywords- Big Data, MapReduce, Hadoop, Schedulers, Execution Speculative

Digital Object Identifier (DOI):http://dx.doi.org/10.18687/LACCEI2018.1.1.326
ISBN: 978-0-9993443-1-6
ISSN: 2414-6390

Estudio comparativo de la estimación del progreso de tareas MapReduce en planificadores especulativos

Apaza Veliz, Ronald Darwin[1], Mamani Aliaga, Alvaro Henry[1]
[1]Universidad Nacional de San Agustín, Arequipa, *ronald.dev1@gmail.com,amamaniali@unsa.edu.pe*

Abstract*– Grandes cantidades de datos se están generando constantemente, por ese motivo surge la necesidad de hacer uso de sistemas paralelos y distribuidos, sin embargo este tipo de sistemas requieren de un cuidadoso control en la gestión de sus recursos, motivo por el cual, surge el modelo de programación MapReduce, el cual haciendo uso de sus planificadores se encarga de gestionar automáticamente los recursos del sistema distribuido, por ejemplo si un nodo del cluster deja de funcionar, MapReduce automáticamente ejecuta la tarea interrumpida en otro ordenador, otro mecanismo similar, es la planificación especulativa, la cual se encarga de detectar aquellas tareas que tienen un tiempo de ejecución anormalmente prologando y ejecuta copias de estas en un ordenador distinto al que se está ejecutando la tarea original, esto se realiza con la esperanza que la tarea de respaldo termine antes que la tarea original y de esta forma se logre reducir el tiempo de ejecución de toda la aplicación, sin embargo los planificadores especulativos actuales presentan algunas limitaciones para calcular correctamente el progreso de las tareas. El cálculo del progreso de las tareas es fundamental para determinar cuando una tarea esta rezagada y una estimación equivocado del progreso podría ocasionar que se desperdicien recursos del sistema y prolongar el tiemplo de ejecución de toda la aplicación, por este motivo en este trabajo se realiza una comparativa para medir la precisión en la estimación del progreso de las tareas de diversos planificadores especulativos y finalmente se señalan algunas pautas que podrían servir de guía para futuras propuestas y que por medio de estas pautas se logre una estimación de progresos más eficiente.

Keywords*—Big Data, MapReduce, Hadoop, Schedulers, Execution speculative, Progress Score.

I. Introducción

En la actualidad grandes cantidades de datos se están generando constantemente, algunos autores señalan que esto se debe al incremento del uso de teléfonos inteligentes, sensores, redes sociales, internet de las cosas, servicios en línea, entre otros [1], un ejemplo de esto es que empresas como Google procesan 20 petabytes de datos al día [2], Facebook procesa más de 500 terabytes al día, Twitter genera más de 12 terabytes al día [3] y los experimentos realizados por el Gran Colisionador de Hadrones generan 60 terabytes de datos al día [4], toda esta información que no puede ser procesada o analizada usando herramientas tradicionales se conoce como *Big Data* [5].

Procesar estas grandes cantidades requiere el uso de sistemas paralelos y distribuidos, sin embargo este tipo de sistemas requieren de una eficiente administración de recursos y un adecuado diseño del sistema [6], por lo tanto si un programador quiere hacer uso de un sistema distribuido, Diaz et al. [7] indican que este este debe encargar de:

- Determinar el modo en que se debe descomponer el problema, para que las tareas más pequeñas se puedan ejecutar en paralelo.
- Asignar tareas a los nodos.
- Asegurarse que los nodos de cómputo reciben la información que necesitan.
- Coordinar la sincronización de los diferentes procesos.
- Establecer las políticas necesarias para compartir los resultados generados.
- Controlar errores de software y fallos de hardware.

A lo largo de los años, diversas herramientas de programación se han desarrollado con el objetivo de eximir al programador de estas responsabilidades, entre las más conocidas se encuentran: OpenMP y MPI (Interfaz de Paso de Mensajes), si bien este tipo de herramientas proporcionan un cierto nivel de abstracción sobre las operaciones de comunicación y sincronización de los nodos, estas aún no cubren por completo los requisitos anteriormente mencionados, esto ocasionaba que gran parte del tiempo para el desarrollo de aplicaciones paralelas sea utilizado para la preparación de la plataforma, sin embargo esto cambió cuando en el año 2004 Google presentó su modelo de programación paralela llamado MapReduce [8], el cual permite procesar grandes conjuntos de datos de forma distribuida [7] y es este modelo que con la ayuda de sus planificadores logran que sea MapReduce el que se encargue de particionar los datos de entrada, la comunicación entre las maquinas, planificar la ejecución de los programas en los diferentes ordenadores y de gestionar los fallos [8], un ejemplo de esto último es que si un ordenador del clúster deja de funcionar MapReduce automáticamente se encarga de volver a ejecutar la tarea fallida en un ordenador distinto al que se está ejecutando la tarea original [10]; otro mecanismo similar a este, es la "planificación especulativa", el cual se encarga de detectar las tareas que tienen un tiempo de ejecución anormalmente prolongado (las cuales son conocidas como *stragglers*) y ejecutar copias de estas en un ordenador distinto al que se están ejecutando la tarea original [9], esto se realiza con el objetivo de reducir el tiempo de ejecución de la aplicación, lo cual se logra, cuando alguna tarea de respaldo logra terminar antes que su tarea original, según Google [8] el planificador especulativo nativo de MapReduce logra disminuir el tiempo de respuesta de las *jobs* (las aplicaciones MapReduce son conocidas como *jobs*) hasta en un 44%, no obstante los planificadores especulativos usados en la

Digital Object Identifier (DOI): http://dx.doi.org/10.18687/LACCEI2018.1.1.326
ISBN: 978-0-9993443-1-6
ISSN: 2414-6390

actualidad aún presentan algunas deficiencias en la estimación del progreso de las tareas, por este motivo en este trabajo se realiza una comparativa de la precisión en la estimación del progreso de tareas MapReduce de los planificadores LATE [8], SAMR [11] y ESAMR [10] frente a un entorno de una creciente cantidad de datos históricos y con una diversidad de aplicaciones históricas.

A. *Trabajos relacionados*

Existen diversos estudios de comparación de planificadores de herramientas de alto desempeño, un ejemplo de ello es el trabajo desarrollado por Mamani et al. [20] en donde se realiza una comparación de algoritmos de planificación de computación en malla.

Por otra parte también se han realizado diversas comparaciones entre los planificadores utilizados por MapReduce, entre los cuales se encuentra el trabajo desarrollado por Naik [12], en el cual se muestra una revisión en donde se examina diversos algoritmos de planificación de trabajos con sus respectivas características, también se estudia el comportamiento los algoritmos LATE, SAMR y ESAMR frente a entornos heterogéneos.

En el trabajo de Thomas [13] se analiza las ventajas y desventajas del planificador especulativo por defecto de Hadoop, LATE, SAMR y ESAMR, también se evalúa la eficacia en de estos para encontrar eficientemente las tareas *straggler*.

En el trabajo de Hadi [14] se estudia los problemas de 15 planificadores de MapReduce y posteriormente se señala las ventajas y desventajas de cada una de ellas.

Todos los trabajos mencionados anteriormente se enfocan en analizar un conjunto de planificadores y posteriormente señalar teóricamente las ventajas y desventajas de estas, en ninguno de los trabajos mencionados se enfocan en la precisión que alcanza cada planificador en la estimación del progreso de las tareas MapReduce, el cual es un proceso de vital importancia para los planificadores especulativos, pues la estimación del progreso es utilizado para identificar cuando una tarea esta rezagada, por lo que una mala estimación de esta, podría conducir a una selección errónea de tareas rezagadas y que las copias de respaldo de iniciadas para estas, solo desperdicien recursos del sistema mientras que las verdaderas tareas rezagadas continuarán prolongando el tiempo de ejecución de toda la aplicación, por este motivo en este trabajo se realiza una comparación de la precisión que alcanzan los planificadores LATE, SAMR y EMSAR en la estimación del progreso de las tareas MapReduce.

B. *Objetivo*

El objetivo principal de este trabajo es realizar una comparativa de la estimación de progresos de las tareas MapReduce de diversos planificadores especulativos y observar el comportamiento de estos cuando son sometidos a diversos cantidades de datos históricos y a diversos tipos de Jobs históricos, posteriormente se busca analizar algunos de los requerimientos necesarios que deben ser considerados para mejorar la precisión en la estimación del progreso de este tipo de tareas.

C. *Organización del trabajo*

Este trabajo está organizando de la siguiente manera: En la sección 2 se muestra la metodología que utilizan diversos planificadores especulativos para estimar el progreso de las tareas MapReduce. En la sección 3 se describe el proceso utilizado para generar los pesos MapReduce sintéticos. En la Sección 4 se muestra la comparación en sus diversos entornos experimentales. En la sección 5 se analiza algunas de las características principales que se deben de tener en cuenta para la formulación de futuras propuestas, finalmente en la sección 6 se muestran las conclusiones obtenidas en este trabajo.

II. ALGORITMOS ESPECULATIVOS

En esta sección se describe la metodología utilizada por los planificadores especulativos LATE, SAMR y ESAMR para estimar el progreso de las tareas MapReduce.

A. *LATE*

Proviene de las siglas de *Longest Approximate Time to End*, es un algoritmo de planificación propuesto por Zaharia et al. [19] en el año 2008 y es el algoritmo de ejecución especulativa usado por Hadoop a partir de la versión 0.21. Sun et al. señala que este planificador intenta mejorar MapReduce, mediante la detección de las verdaderas tareas *stragglers*, en este trabajo se propone que las tareas con el tiempo restante más prolongado, sean las tareas que deben ser especuladas, el procedimiento seguido por esta propuesta consiste en calcular el progreso de las tareas (*PS*), para ello se hace uso del número de pares clave/valor que necesitan ser procesados *N*, la cantidad de pares clave/valor que ya han sido procesados *M* y en el caso de las tareas *reduce*, también se requiere la cantidad de fases que esta haya finalizado (*K*), el cálculo del progreso se muestra en (1) para el caso de las tareas *map* y (2) en el caso de las tareas *reduce*, posteriormente es necesario calcular la tasa de progreso de la tarea (*PR*), esta se calcula utilizando la ecuación (3), en donde *Tr* es la cantidad de tiempo (en segundos) que se ha ejecutado la tarea evaluada, una vez establecido la tasa de progreso se calcula el tiempo restante de la tarea *TTE* utilizando (4).

$$PS = M / N \quad (1)$$

$$PS = 1/3 * (K + M / N) \quad (2)$$

$$PR = PS / Tr \quad (3)$$

$$TTE = (1 - PS) / PR \qquad (4)$$

Chen et al. [11] afirma que aunque LATE usa una eficiente estrategia para detector tareas, *stragglers*, este algunas veces inicia tareas especulativas de forma incorrecta y esto se debe a que esta propuesta no logra estimar correctamente el tiempo restante de las tareas, debido a que los pesos de las fases de las tareas *map* y *reduce* son asignados de forma estática, los valores 1 y 0 son asignados a las fases de las tareas *map* y los valores 0.33, 0.33 y 0.34 son asignados a las tareas *reduce*, esta asignación estática ocasiona una estimación incorrecta del progreso de las tareas, la cual es necesaria para calcular el tiempo restante de las mismas, por ejemplo si se desea calcular el progreso de una tarea *reduce* que ha terminado su primera fase (fase de copiado), este planificador consideraría que la tarea tiene un progreso del 33%, sin embargo esta estimación puede ser errónea, pues como lo indica Dean & Ghemawat [8] es habitual que la primera fase de las tareas *reduce* utilicen una mayor cantidad de tiempo (en comparación a las otras 2 fases) y a que esta fase involucra la comunicación de datos a través de la red, lo cual suele ser un proceso lento.

B. *SAMR*

De las siglas de *Self-Adaptative MapReduce scheduling algorithm* [11] es un algoritmo de planificación, el cual calcula el progreso de las tareas dinámicamente y se adapta continuamente a diversos entornos de forma automática. Al igual que el algoritmo LATE, este planificador identifica las tareas lentas estimando el tiempo de ejecución restante, sin embargo este no asigna pesos estáticos a las fases de las tareas, en su lugar SAMR ajusta el peso de las fases de acuerdo a su información histórica (previamente almacenada); esto con la finalidad de obtener una mayor precisión en la estimación del progreso de las tareas.

Sun et al. [10] afirma que debido a que SAMR utiliza la información histórica guardada en cada nodo para modificar el peso de las fases, este planificador adopta pesos más realistas que los establecidos de forma estática, por lo tanto comparado con el planificador por defecto de Hadoop y el planificador LATE, el planificador SAMR trabaja mejor, especialmente en entornos heterogéneos.

No obstante Li et al. [15] afirma que este planificador no resuelve el problema crucial ya que solamente considera la heterogeneidad del hardware; y no considera que diferentes tipos de tareas pueden ser procesadas en un mismo nodo y que por lo tanto las fases de las mismas podrían tener diferentes pesos; tal y como se demuestra en las figuras 1 y 2 en donde se muestra que existen diferentes pesos de fase dependiendo al tipo de *job* que se está procesando.

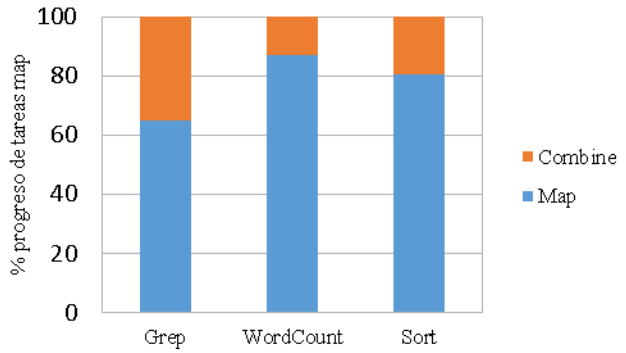

Fig 1. Variabilidad de los pesos de las fases en las tareas map, según el tipo de algoritmo, imagen adaptada de [16]

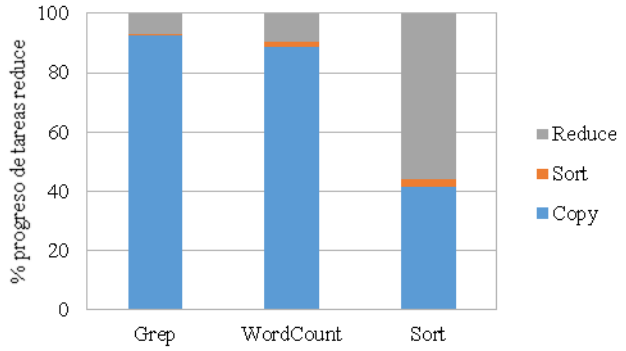

Fig. 2. Variabilidad de los pesos de las fases en las tareas reduce, según el tipo de algoritmo, imagen adaptada de [16]

C. *ESAMR*

Es una mejora de SAMR [11], que proviene de las siglas de *Enhanced Self Adaptative MapReduce* [10], esta propuesta toma en consideración que además de la heterogeneidad de hardware, existen muchos otros factores que pueden afectar el peso de las fases de las tareas, por este motivo ESAMR propone utilizar el algoritmo de agrupamiento K-Medias para agrupar la información histórica y de esta forma obtener grupos de tareas con características similares, logrando así que cuando se requiera asignar los pesos de las fases, solo se tome en cuenta la información proporcionada al grupo al cual pertenece la tarea evaluada y por lo tanto se le asigne pesos más realistas.

Sin embargo Apichanukul et al. [9] señala que agrupar la información histórica incrementa una sobrecarga en el periodo de procesamiento del *job* y que además es difícil asignar el número adecuado de grupos al algoritmo K-Means en un entorno en donde se planea ejecutar múltiples *jobs*.

III. Generando los pesos sintéticos

Antes de realizar las comparativas fue necesario generar pesos sintéticos, para ello se modificó el código fuente de Hadoop 0.21 y se almaceno los pesos correspondiente a la ejecución de

50 aplicaciones; la cantidad de tareas generadas y los tipos de *jobs* que se ejecutaron son mostrados en la tabla 1.

Tabla 1
Jobs ejecutados

Job	Promedio de tareas map	Promedio de tareas reduce	Cantidad de ejecuciones
Grep	154	2	10
Pi	500	1	10
Sort	150	1	10
Validate	1	1	10
Wordcount	153	1	10

Se utilizó como referencia el promedio de los pesos de los *jobs* mostrados en la tabla 1, junto con los pesos promedio obtenidos en los trabajos de Sun [17], Xu et al. [18] y Chen et al. [16], la recopilación de estos *jobs* de referencia con sus respectivos pesos promedio son mostrados en la tabla 2, en donde los pesos de las tareas map son representados por *M1* y *M2* y los pesos de las tareas reduce por *R1*, *R2* y *R3*.

Tabla 2
Recopilación de pesos de fase promedio de diversos *jobs*

Job	M1	M2	R1	R2	R3
Grep	0.920	0.080	0.520	0.070	0.410
	0.648	0.352	0.925	0.008	0.067
	0.997	0.003	0.581	0.002	0.417
K-means	0.880	0.120	0.540	0.020	0.440
Pi	0.996	0.004	0.998	0.000	0.002
Sort	0.090	0.910	0.700	0.030	0.270
	0.806	0.194	0.415	0.027	0.558
	0.823	0.177	0.801	0.089	0.110
Validate	0.999	0.001	0.283	0.007	0.710
Wordcount	0.210	0.790	0.370	0.020	0.610
	0.900	0.100	0.470	0.110	0.420
	0.870	0.130	0.888	0.015	0.097
	0.911	0.089	0.988	0.001	0.011

Para establecer los rangos entre los que debe oscilar cada peso se calculó la diferencia entre el peso promedio y los pesos máximos y mínimos de cada fase (omitiendo los valores atípicos), los resultados obtenidos se muestran en la tabla 3, en donde el primer valor representa la diferencia máxima que pueden tener los pesos con un valor menor al peso promedio y el segundo valor representa la diferencia máxima que pueden tener aquellos pesos generados con un valor mayor al del promedio.

Tabla 3
Diferencia mínima y máxima permita, respecto a los pesos promedio

	Grep	Pi	Sort	Validate	Wordcount
M1	0.025	0.009	0.219	0.001	0.502
	0.002	0.003	0.146	0.001	0.049
M2	0.002	0.003	0.146	0.001	0.049
	0.025	0.009	0.219	0.001	0.502
R1	0.429	0.001	0.021	0.022	0.001
	0.417	0.001	0.046	0.033	0.001
R2	0.002	0.000	0.060	0.002	0.001
	0.003	0.000	0.029	0.005	0.000
R3	0,415	0.001	0.012	0.034	0.001
	0.426	0.001	0.014	0.024	0.001

En el caso del algoritmo K-means, no se dispone de un listado de pesos que nos ayuden a calcular las diferencias máximas y mínimas, por este motivo para generar los pesos sintéticos para este *job*, se usó las diferencias promedio de todos los *jobs* de la tabla 3.

IV. Experimentos

Para realizar los experimentos se implementó los algoritmos de estimación de progreso utilizadas en los planificadores LATE, SAMR y ESAMR, para calcular la precisión alcanzada por cada uno de los algoritmos implementados, se calculó el porcentaje de precisión que representa el progreso estimado por cada planificador respecto al progreso real de las tareas evaluadas.

En este trabajo se mide la precisión en 2 casos, para el primer caso se generaron pesos sintéticos correspondiente a los *jobs* mostrados en la tabla 2, posteriormente se usaron esos pesos sintéticos para evaluar el impacto de la precisión de los planificadores a medida que se incrementa la cantidad de datos históricos, para el segundo caso se generaron pesos sintéticos utilizando pesos promedio aleatorios, esto con la finalidad de simular un entorno real, en donde se podrían ejecutar diversos *jobs*, posteriormente se evalúa el impacto en la precisión de los planificadores cuando se tiene una mayor diversidad de Jobs.

A. Midiendo la precisión según la cantidad de datos históricos.

Para observar el impacto de la precisión de los planificadores según la cantidad de datos históricos disponibles, se generaron *jobs* con pesos sintéticos utilizando los pesos promedio que se muestran en la tabla 2, una vez obtenidos los jobs sintéticos, se seleccionó aleatoriamente un grupo de *jobs*, para ser parte de los datos históricos, asegurándonos este grupo contenga al menos un *job* de cada *job* de referencia (tabla 2), para evaluar la precisión en la estimación del progreso se utilizó 6 *jobs* (uno por cada *job* de referencia), en las figuras 3 y 4 se muestra la precisión alcanzada por cada planificador de acuerdo a la cantidad de *jobs* históricos.

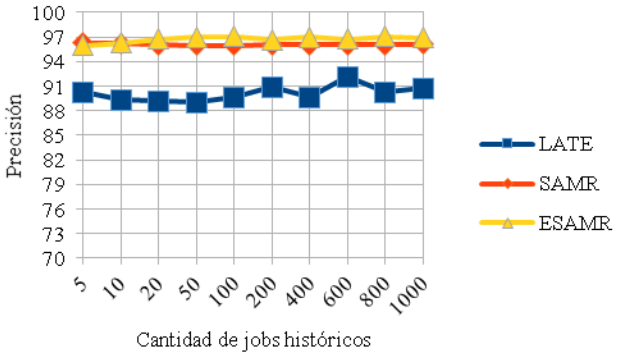

Fig 3. Comparativa de precisión en la estimación del progreso de tareas map según la cantidad de datos históricos

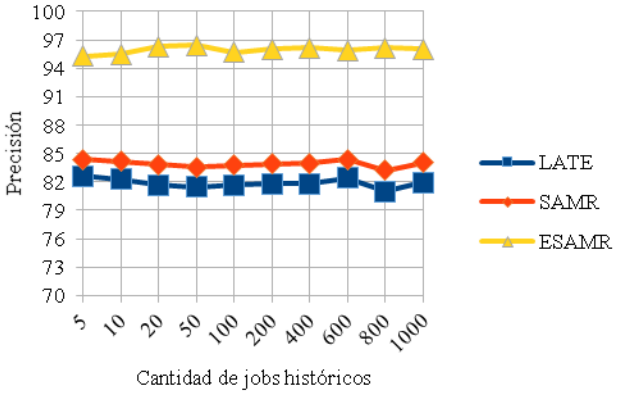

Fig 4. Comparativa de precisión en la estimación del progreso de tareas reduce según la cantidad de datos históricos

B. *Entorno con múltiples tipos de jobs*

Una de las desventajas de ESAMR es que el algoritmo de agrupamiento usado por este requiere que se le asigne a priori el número grupos, el cual es un valor difícil de calcular en entornos en donde se planea ejecutar múltiples *jobs*, por tal motivo, para verificar la precisión que ESAMR alcanza en entornos de múltiples *jobs* se generaron *jobs* con pesos de referencia aleatorios, posteriormente se midió la precisión alcanzada por cada planificador de acuerdo a la cantidad de *jobs* históricos aleatorios, los resultados de este experimento son mostrados en las figuras 5 y 6.

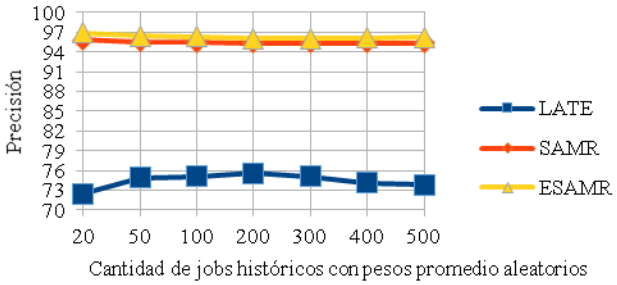

Fig 5. Comparativa de precisión en la estimación del progreso de tareas map según la diversidad de jobs

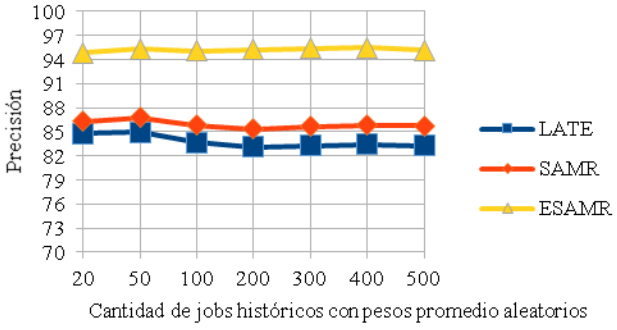

Fig 6. Comparativa de precisión en la estimación del progreso de tareas reduce según la diversidad de jobs

Como se observa en las figuras 5 y 6 la precisión alcanzada por los planificadores SAMR no sufren de un gran deterioro cuando se ejecutan jobs de distintos tipos, incluso ESAMR que utiliza como máximo 20 grupos. En su agrupamiento mantiene una precisión similar a medida que se incrementa la cantidad de Jobs aleatorios.

C. *Midiendo el tiempo de ESAMR*

Una de las desventajas de ESAMR, es que este ocasiona una ligera sobrecarga de tiempo al *job*, esto se debe a que está propuesta requiere agrupar los pesos históricos para calcular los pesos de las fases de las tareas MapReduce, por este motivo en esta sección se muestra el tiempo que requiere el agrupamiento de los pesos de las tareas a medida que se va incrementando la cantidad de Jobs históricos, los resultados de este experimento se muestran en la figura 7.

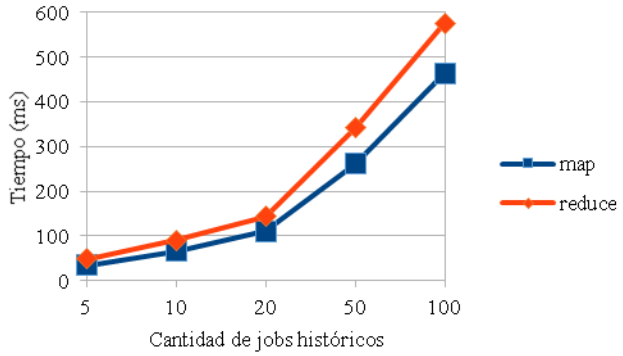

Fig 7. Comparativa del tiempo requerido por ESAMR para agrupar los pesos históricos map y reduce

Como se observa en las figura 7 a medida que se incrementa la cantidad de *jobs* históricos, también se incrementa el tiempo requerido por ESAMR y esto se debe a que ESAMR agrupa todos los pesos históricos disponibles, incluso cuando estos ya fueron agrupados en la ejecución de un *job* anterior.

V. Análisis de Resultados

Los resultados obtenidos en este trabajo muestran que utilizar una técnica de agrupamiento mejora la precisión en la estimación de progresos de las tareas MapReduce.
Además en los experimentos 3 y 4 se observa que la cantidad de *jobs* históricos no tiene una repercusión significativa en la precisión de SAMR y ESAMR y esto se debe los pesos pertenecientes a un mismo *job* poseen un alto grado de similitud, lo cual ocasiona que pese a que se dispone de una mayor cantidad de datos históricos, estos no influyan en la precisión de SAMR y ESAMR.

Por otra parte una de las desventajas de ESAMR señalada por Apichanukul [9] es que es difícil asignar un número de centroides adecuados en entornos donde se planea ejecutar múltiples jobs, sin embargo en los experimentos 5 y 6 se muestra que incluso cuando se ejecutan múltiples tipos de *jobs* la precisión de esta se mantiene, lo cual indica que los 20 grupos establecidos en ESAMR permite estimar el progreso de las tareas MapReduce con una precisión entre el 96% al 97% en el caso de las tareas *map* y un 94.8% y 95.5% para el caso de las tareas *reduce*.

Finalmente en los resultados mostrados en la figura 7, se muestra que efectivamente existe una ligera sobrecarga en la estimación del progreso de las tareas MapReduce y esto se debe a que es necesario agrupar previamente los datos históricos antes de la ejecución de cada *job*, además se observa que en el caso de ESAMR esta sobrecarga se incrementa a medida que se incrementa la cantidad de datos históricos.

VI. Análisis del agrupamiento de pesos MapReduce

Como se mencionó en las sección anterior ESAMR alcanza una mayor precisión en la estimación de progresos que otros planificadores como SAMR y LATE, no obstante, autores como Naik et al. [12] indican que ESAMR podría ser mejorado utilizando una técnica de agrupamiento más adecuada, por dicho motivo en esta sección se detallan algunos de las características que se deberían tener en cuenta en la agrupación de pesos MapReduce.

Primero, los datos que se pretende agrupar son los pesos de las tareas MapReduce, los cuales son valores comprendidos entre 0 y 1, por lo tanto el algoritmo seleccionado debería ser efectivo agrupando valores numéricos.

Una vez agrupados los pesos históricos, se utiliza los centroides de cada grupo para asignar los pesos de las futuras tareas, esto significa que el centroide de cada grupo debe ser el punto más representativo del grupo, por tal motivo el algoritmo seleccionado debe ser capaz de agrupar lo que se conoce como grupos basados en su centro.

Dado que todos los pesos de los *jobs* son almacenados como pesos históricos, es posible que existan valores atípicos y estos deberían ser controlados para que no deterioren la calidad del agrupamiento.

Debido a que en un entorno real, se podrían ejecutar distintos tipos de aplicaciones, el algoritmo de agrupamiento seleccionado debería prescindir que se le asigne una cantidad fija de grupos.

Finalmente con el objetivo de reducir la sobrecarga ocasionada por el algoritmo de agrupamiento seleccionado se debería utilizar una técnica con un bajo coste computacional y la sobrecarga reduciría aún más si el algoritmo utilizado, permite el agrupamiento de flujos de datos, es decir, que el algoritmo permita agregar los datos nuevos a los agrupamientos previamente realizados y no volver a agrupar todos los datos históricos disponibles desde la ejecución del primer *job*.

VII. Conclusiones

En este trabajo se realiza una comparativa de la estimación del progreso de los planificadores especulativos LATE, SAMR y ESAMR, estos fueron comparados en 2 entornos diferentes, en el primer entorno se observó el comportamiento de los planificadores respecto a la cantidad de *jobs* históricos, en esta se observó que SAMR y ESAMR mantienen aproximadamente la precisión sin importar la cantidad de datos históricos que se disponga.

En el segundo entorno se observó el comportamiento de los planificadores cuando estos tiene como datos históricos una gran variedad de *jobs*, en el caso de SAMR mantiene una precisión similar en el caso de las tareas map y se incrementa ligeramente la precisión en el caso de las tareas reduce, por otra parte ESAMR baja ligeramente la precisión (aproximadamente en un 1%) en el caso de las tareas map y reduce, finalmente se analizó algunas de las características que se deben tener en consideración para futuras propuestas de mejora en la estimación del progreso de las tareas MapReduce.

Para un agrupamiento adecuado de pesos MapReduce, se recomienda utilizar el algoritmo de agrupamiento BIRCH [21], el cual permite encontrar grupos basados en su centro, lo cual es un requisito fundamental para asignar nuevos pesos MapReduce, además BIRCH posee un bajo costo computacional (*O(N)*), por lo que la sobrecarga originada en el agrupamiento será menor que el propuesto en ESAMR.

Agradecimientos

Se agradece a CONCYTEC por el apoyo financiero otorgado al autor Ronald Apaza.

Referencias

[1] Khan, Mukhtaj, “Hadoop performance modeling and job optimization for big data analytics”, 2015.

[2] Sagiroglu, Seref, and Duygu Sinanc. "Big data: A review." Collaboration Technologies and Systems (CTS), 2013 International Conference on. IEEE, 2013.

[3] Zaslavsky, Arkady, Charith Perera, and Dimitrios Georgakopoulos. "Sensing as a service and big data.", 2013.

[4] Chen, CL Philip, and Chun-Yang Zhang. "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data." Information Sciences 275 (2014): 314-347.

[5] Pérez, María. Big data-Técnicas, herramientas y aplicaciones. Alfaomega Grupo Editor, 2015.

[6] Lopes Bezerra, Aprigio Augusto, and Porfidio Hernández Budé. "Planificación de trabajos en clusters hadoop compartidos.", 2015.

[7] Díaz Cañizares, José Fidel, and Porfidio Hernández Budé. "Integración de Hadoop con planificadores batch.", 2011.

[8] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[9] Apichanukul, Worachate, Jun Kawahara, and Shoji Kasahara. "Accuracy Improvement for Backup Tasks in Hadoop Speculative Algorithm." Computer and Information Technology (CIT), 2016 IEEE International Conference on. IEEE, 2016.

[10] Sun, Xiaoyu, Chen He, and Ying Lu. "ESAMR: An enhanced self-adaptive mapreduce scheduling algorithm." Parallel and Distributed

Systems (ICPADS), 2012 IEEE 18th International Conference on. IEEE, 2012.
[11] Chen, Quan, et al. "Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment." Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on. IEEE, 2010.
[12] Naik, Nenavath Srinivas, Atul Negi, and V. N. Sastry. "A review of adaptive approaches to MapReduce scheduling in heterogeneous environments." Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on. IEEE, 2014.
[13] Thomas, Liya, and R. Syama. "Survey on MapReduce scheduling algorithms." International Journal of Computer Applications 95.23 (2014).
[14] Yazdanpanah, Hadi. "Scheduling Algorithms for MapReduce Framework.", 2015
[15] Li, Yuanzhen, et al. "A new speculative execution algorithm based on C4. 5 decision tree for Hadoop." International Conference of Young Computer Scientists, Engineers and Educators. Springer, Berlin, Heidelberg, 2015.
[16] Chen, Qi, Cheng Liu, and Zhen Xiao. "Improving MapReduce performance using smart speculative execution strategy." IEEE Transactions on Computers 63.4 (2014): 954-967.
[17] Xiaoyu Sun, "An enhacend self-adaptive MapReduce scheduling algorithm", 2012.
[18] Zhao, Xu, et al. "A parameter dynamic-tuning scheduling algorithm based on history in heterogeneous environments." ChinaGrid Annual Conference (ChinaGrid), 2012 Seventh. IEEE, 2012.
[19] Zaharia, Matei, et al. "Improving MapReduce performance in heterogeneous environments." Osdi. Vol. 8. No. 4. 2008.
[20] Mamani-Aliaga, Alvaro H., Alfredo Goldman, and Yanik Ngoko. "A comparative study on task dependent scheduling algorithms for grid computing." Computer Systems (WSCAD-SSC), 2012 13th Symposium on. IEEE, 2012.
[21] Zhang, T., Ramakrishnan, R., & Livny, M. (1996, June). BIRCH: an efficient data clustering method for very large databases. In ACM Sigmod Record (Vol. 25, No. 2, pp. 103-114). ACM.