Desarrollo de algoritmos genéticos utilizando diferentes frameworks de MapReduce: MPI vs. Hadoop

Juan José Barbero, Martín Tamagusku, Hugo Alfonso[1],
Carlos Bermudez, Gabriela Minetti[1], Carolina Salto[1]
Laboratorio de Investigación en Sistemas Inteligentes (LISI)
Facultad de Ingeniería - Universidad Nacional de La Pampa
Calle 110 Esq. 9 (6360) General Pico - La Pampa - Rep. Argentina
Te. / Fax: (02302) 422780/422372, Int. 6302
e-mail: [1]{minettig, saltoc, alfonsoh@ing.unlpam.edu.ar}

Resumen MapReduce es un paradigma popular, que permite a los usuarios no especializados utilizar grandes plataformas computacionales paralelas de manera transparente. Hadoop es la implementación más utilizada de este paradigma y, de hecho, para una gran cantidad de usuarios, la palabra Hadoop y MapReduce son intercambiables. Pero, hay otros framewoks que implementan este paradigma de programación, como MapReduce-MPI. Dado que las técnicas de optimización pueden beneficiarse enormemente de este tipo de modelado informático de uso intensivo de datos, en esta línea de investigacón analizamos el efecto del rendimiento del desarrollo de algoritmos genéticos (GA) utilizando diferentes marcos de MapReduce (MRGA).

Palabras claves: Metaheurísticas, optimización, operadores

Contexto

Esta línea de investigación se desarrolla en el marco del proyecto de investigación llevado a cabo en el Laboratorio de Investigación de Sistemas Inteligentes (LISI), de la Facultad de Ingeniería de la Universidad Nacional de La Pampa, acreditado por dicha facultad y dirigido por la Dra. Minetti. Cabe destacar que desde hace varios años, los integrantes de estos proyectos mantienen una importante vinculación con investigadores de la Universidad Nacional de San Luis (Argentina) y de la Universidad de Málaga (España), con quienes se realizan publicaciones conjuntas. En particular, la línea de investigación aquí descripta se lleva adelante conjuntamente con los Dres. Alba y Luque de la Universidad de Málaga.

Introducción

MapReduce (MR) es un paradigma de programación, creado por los investigadores de Google [1], para procesar y generar grandes conjuntos de datos con un algoritmo paralelo y distribuido en clusters y en la nube. El programador puede abstraerse de los problemas de la programación distribuida y paralela, ya que MR administra el equilibrio de carga, el rendimiento de la red y la tolerancia a fallas en algunos frameworks. Esto hizo que MR fuera popular para aquellos que no sabían sobre programación paralela pero que querían usarla, creando una nueva rama del paralelismo donde el enfoque está en la aplicación y no en la explotación del hardware subyacente.

La implementación de MR en Google es una librería propietaria en C++, con comunicación entre máquinas en red a través de llamadas a procedimientos remotos. Permite la tolerancia a fallas cuando se emplean grandes cantidades de máquinas, y puede usar discos como memoria fuera del núcleo para procesar conjuntos de datos a escala de petabytes [2]. HADOOP incorpora también estas características en su implementación [3], pero es una librería de código abierto en Java

que, además, soporta varios lenguajes de programación.

Puesto que que Hadoop permite el paralelismo de datos y control, es una ciencia común preguntarnos sobre otras herramientas de software que realicen trabajos similares. MapReduce-MPI (MR-MPI) [4] es una librería construida sobre MPI con un objetivo similar. Al tratarse de una librería de código abierto en C++, permite un control preciso sobre la memoria y el formato de los datos asignados por cada procesador durante una operación de MapReduce. Tanto *map* como *reduce* son funciones especificadas por el usuario que se pueden escribir en los lenguajes C, C++ y Python. Aquí se puede tener más control sobre la plataforma, lo que permite mejorar el rendimiento del ancho de banda y reducir los costos de latencia. Si bien sus servicios (para Hadoop y MR-MPI) no son exactamente iguales, la mayoría de las aplicaciones se pueden escribir sobre ellas, y se hace un análisis comparativo para cualquier científico curioso, a fin de ofrecer evidencias sobre su desempeño.

Dada la necesidad científica e industrial de nuevos algoritmos evolutivos, en particular de los algoritmos genéticos (GA) escalables, un modelo informático de uso intensivo de datos puede beneficiar enormemente el campo para construir nuevos modelos de optimización y aprendizaje automático con ellos. Por esta razón, el objetivo de esta línea de investigación es analizar diferentes formas de introducir los elementos de MR en un algoritmo genético simple (SGA) [5]. Los primeros resultados de este trabajo se cristalizan en una técnica llamada MRGA [6], que surge de una paralelización de cada iteración de un SGA bajo el Modelo Paralelo de Nivel de Iteración [7].

Desarrollo

En esta sección se describe la primera parte del estudio propuesto en esta línea de investigación. Para ello es necesario realizar una breve descripción del paradigma MR.

El paradigma MapReduce organiza una aplicación en un par (o una secuencia de pares) de las funciones *map* y *reduce* [1]. Las funciones *map* son independientes entre sí y se ejecutan en paralelo sin conexión entre ellas. Las segundas unen los valores con la misma clave.

Cada función *map* toma como entrada un conjunto de pares clave/valor (registros) de los archivos de datos y genera un conjunto de pares clave/valor intermedio. Luego, MR agrupa todos estos valores intermedios asociados con una misma clave. Un grupo de valores y su clave asociada es la entrada a la función *reduce*, que combina estos valores para producir un conjunto nuevo y posiblemente más pequeño de pares clave/valor que se guardan en archivos de datos. Además, esta función recibe los valores intermedios a través de un iterador, lo que permite al modelo manejar listas de valores que son demasiado grandes para formar parte de la memoria principal.

Los datos de entrada se dividen automáticamente en un conjunto de M divisiones cuando las invocaciones *map* se distribuyen en múltiples máquinas, donde cada división de entrada se procesa mediante una invocación *map*. El espacio de claves intermedias se divide en R piezas, por medio de la función de partición (por ejemplo, $hash(key) \mathbf{mod} R$) que se distribuyen en R invocaciones de *reduce*.

MRGA se basa en el SGA propuesto por [5], que solo utiliza los operadores de selección y recombinación. El algoritmo propuesto preserva el comportamiento de SGA, pero recurre a la paralelización de algunas partes, tales como: la evaluación y la aplicación de operadores genéticos. Si bien, MRGA realiza varias operaciones en paralelo, su comportamiento es el mismo que el de un GA secuencial.

Se utiliza un par clave/valor para representar a individuos en la población. La representación de un individuo para el problema de la mochila es una secuencia de bits, una para cada elemento, que indica si el elemento está en la mochila o no. A continuación, comienza el proceso evolutivo iterativo, pero el modelo MR no ofrece métodos para implementar algoritmos iterativos. En consecuencia, se genera una cadena de múltiples tareas MapReduce utilizando la salida de la última como la entrada de la siguiente. Una consideración importante es que la condición de terminación debe computarse dentro del proceso principal del programa MapReduce. Cada iteración consta de las funciones *map* y *reduce*.

Teniendo en cuenta los problemas de gran tamaño, los individuos grandes y las funciones de evaluación complejas, las funciones *map* calculan la aptitud de los individuos. Como a cada *map*, MR le ha asignado una porción diferente de datos, se evalúa un conjunto de diferentes individuos en paralelo. Esta aptitud se agrega como valor en el par clave/valor.

Las funciones *reduce* llevan a cabo las operaciones genéticas. La selección del torneo binario se realiza localmente con el espacio clave intermedio, que se distribuye en la etapa de partición después de la operación *map*. El operador de cruce uniforme (UX) se aplica sobre los individuos seleccionados. El reemplazo generacional se implementa al escribir estos nuevos individuos en un archivo para convertirse en la entrada del *map* en la próxima tarea MapReduce (una nueva iteración).

Resultados obtenidos

En este primera etapa, se estudian diferentes formas de introducir los elementos MapReduce en un algoritmo genético simple. Como consecuencia, surgieron dos implementaciones de MRGA utilizando ambos frameworks de código abierto MapReduce, Hadoop y MR-MPI, para generar los algoritmos MRGA-H y MRGA-M respectivamente. Estos algoritmos resuelven un problema de gran tamaño de interés industrial como es el problema de la mochila. Cuando se analiza la eficiencia, detectamos que los dos algoritmos MRGA necesitan el mismo número de iteraciones para encontrar sus mejores soluciones. La calidad de las soluciones obtenida por los algoritmos es bastante similar, observándose una pequeña ventaja a favor de MRGA-M. Este es un resultado esperado porque ambos algoritmos MRGA se basan en el mismo algoritmo genético selecto-recombinativo.

Considerando la escalabilidad algorítmica desde el punto de vista de la dimensión del problema, observamos que MRGA-H es más escalable que MRGA-M. Las razón de esto es que el crecimiento del tiempo requerido por MRGA-H no es directamente proporcional al incremento en la dimensión de la instancia, es decir, el porcentaje de crecimiento del tiempo de ejecución disminuye a medida que aumenta el número de elementos.

Formación de recursos humanos

Cada año se incorporan al proyecto alumnos avanzados en la carrera Ingeniería en Sistemas, quienes trabajan en temas relacionados a la resolución de problemas de optimización usando técnicas inteligentes, con el objeto de guiarlos en el desarrollo de sus tesinas de grado y, también, de formar futuros investigadores científicos. Por otra parte, los docentes-investigadores que integran el proyecto realizan diversos cursos de posgrado relacionados con la temática del proyecto, con el objetivo de sumar los créditos necesarios para cursar carreras de posgrado.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *OSDI'04: Proceedings of the 6TH Conference on Symposium on Operating Systems Desing and Implementation*. USENIX Association, 2004.

[2] ——, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[3] "Welcome to Apache™ Hadoop®!" The Apache Software Foundation, http://hadoop.apache.org/, Tech. Rep., 2014.

[4] S. Plimpton and K. Devine, "Mapreduce in MPI for large-scale graph algorithms." *Parallel Computing*, vol. 37, no. 9, pp. 610–632, 2011.

[5] D. Goldberg, *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, 2002.

[6] C. Salto, G. F. Minetti, E. Alba, and G. Luque, "Developing genetic algorithms using different mapreduce frameworks: MPI vs. hadoop," in *CAEPIA*, ser. Lecture Notes in Computer Science, vol. 11160. Springer, 2018, pp. 262–272.

[7] E. Talbi, *Metaheuristics: From Design to Implementation*. Wiley Publishing, 2009.