ANÁLISIS DEL RENDIMIENTO DE UN ALGORITMO GENÉTICO PARALELO BASADO EN SPARK Y UN ALGORITMO GENÉTICO PARALELO BASADO EN PTHREADS PARA LA GENERACIÓN DE PAIRWISE TEST SUITE

Ruth Huillca-[1] Melvin Salcedo-[1] Gabriela Rojas[1]

UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

16 julio 2018

Contenido

Abstract

El pairwise es una técnica efectiva para obtener las situaciones de pruebas requeridas para llegar a cierta cobertura , el objetivo del pairwise es cubrir todas las posibles combinaciones de dos factores. Los algoritmos genéticos han sido la herramiena de uso que muchos investigadores han usado para la generación de conjuntos de prueba por pares.Pero a pesar de la optimización que ofrecen los algoritmos genéticos este es un proceso que aún lleva mucho tiempo ,y esto conduce a limitaciones para el uso de algoritmos genéticos hacia problemas de prueba a gran escala. Aplicar paralelismo nos permitirá mejorar el rendimiento de cómputo y así también mejorar la calidad de soluciones.En este documentos realizamos el análisis del algoritmo genético paralelo basado en spark y ptreads,para la plataforma spark utilizamos una técnica que aborda dos fases una es la paralelización evaluación de aptitud y la paralelización de operación genética y la librerá pthreads para la otra forma de paralelización del algoritmo gnético .Como resutados de nuestros experimentos efectivamente comprobamos que el algoritmo genético paralelo basado en spark es mejor respecto a algoritmo genético secuencial y también respecto al algoritmo basado en pthreads ,el algoritmo basado en spark muetra mejor rendimiento computacional .

Spark

Apache Spark combina un sistema de computación distribuida a través de clusters de ordenadores con una manera sencilla y elegante de escribir programas. Apache Spark mejora con respecto a los demás sistemas en cuanto a la computación en memoria. RDD permite a los programadores realizar operaciones sobre grandes cantidades de datos en clusters de una manera rápida y tolerante a fallos.

Threading de Python

Es una biblioteca relacionada con Python para la programación de soluciones que emplean múltiples CPU o CPU multinúcleo en un entorno de multiprocesamiento simétrico (SMP) o memoria compartida, o potencialmente un gran número de computadoras en un clúster o entorno de grillas.

Algoritmo paralelo en spark

La evaluación del fitness es la primera fase de paralelización que incluye las etapas :La etapa 1 genera la población inicial que luego se paraleliza en diferentes particiones de RDD en la etapa 2. El valor del fitness de cada individuo se evalua en diferentes particiones desde la etapa 2 hasta la etapa 3.Las operaciones genéticas se realizan en paralelo desde la etapa 4 a la etapa 5. La etapa 6 recoge los mejores individuos de diferentes particiones.

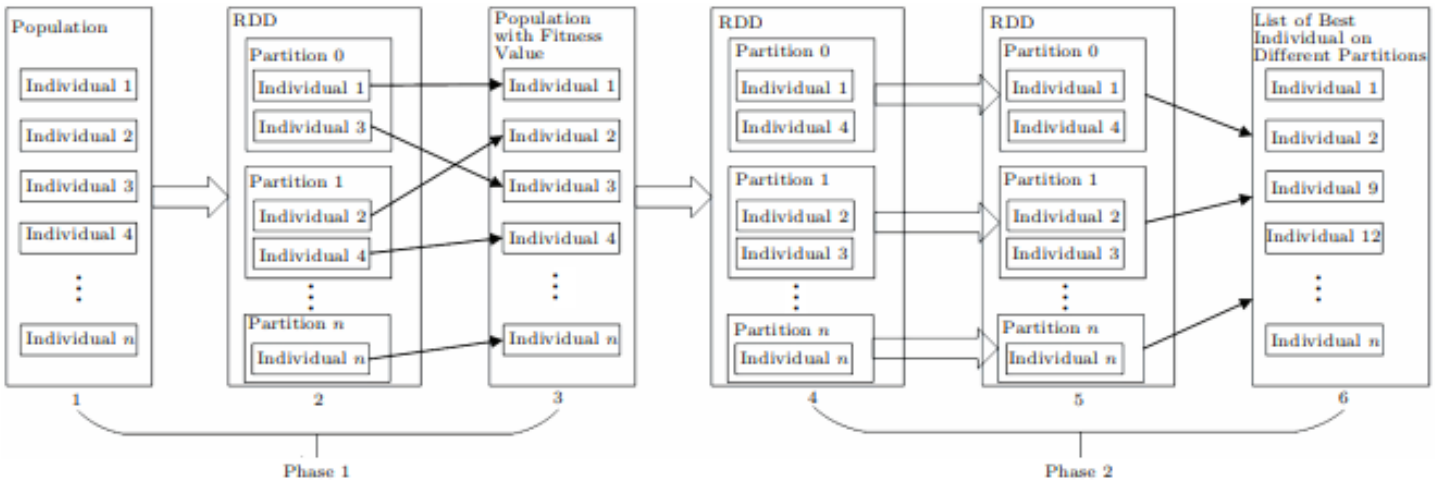

FIGURE – Arquitectura

Pairwise Testing

El Pairwise Testing es una técnica básica obtener las situaciones de prueba requeridas para llegar a una cierta cobertura. El objetivo del Pairwise Testing es probar todas las posibles combinaciones de dos factores. Esto redunda en una gran reducción en el número de casos de prueba, obteniéndose aún así buenos resultados en la detección de defectos.

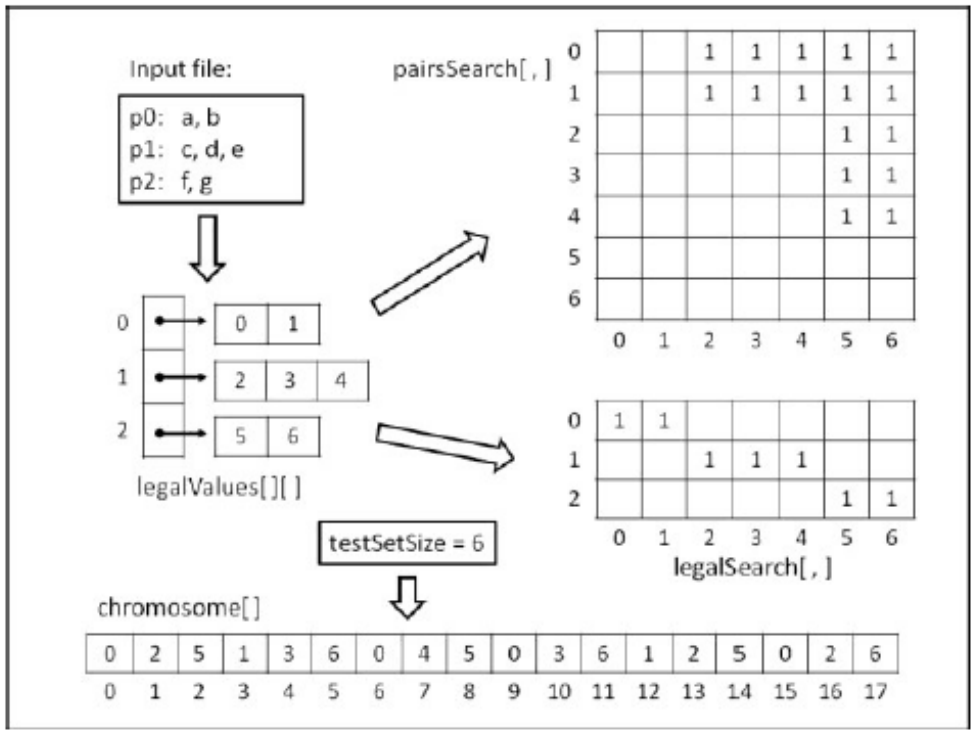

FIGURE – Pairwise

Algoritmos

Paralelización en dos fases.

- Entradas el archivo de texto de valores de parámetros, el número de parámetros, el número de valores para cada parámetro, el tamaño del conjunto de pruebas y el tamaño de población deseado.
- El resultado es un conjunto de pruebas casi mınimo con una cobertura del 100 por ciento por pares
- Se genera el número de todos los pares a cubrir. La población inicial(individuos de tamaño natural), se componen de m casos de prueba que se crean aleatoriamente seleccionando cada espacio de manera uniforme entre todos los valores posibles.
- Paraleliza la primera fase. Luego, los pares clave-valor recopilados por el conductor se ordenan por el fitness.
- Si el valor del primer par es AP, su clave es el mejor individuo que se devolverá. De lo contrario, entra en la paralelización de la segunda fase.

Algoritmos

- La lista de pares(clave, valor), clave(individuo, fitness) y el valor es la cantidad de generación necesaria para encontrar una solución, se ordena por el fitness y el número de generación. Si el valor del primer par es AP, su clave es el mejor individuo que se devolverá.

Input: *pv.txt:* parameter-values text file
k: number of parameters in SUT
v_i: number of values for each parameter
m : test suite size
$popsize$: desired population size
Output: a pairwise test suite that covers all pairs of values at least once

1: $AP \leftarrow getNumOfAllPairs(k, v_i)$
2: $Population \leftarrow initializePop(m, popsize, \text{“}pv.txt\text{”})$
3: $populationRDD \leftarrow parallelize(Population)$
4: **For** each individual $I_j \in populationRDD$ **do**
5: $fitnessRDD \leftarrow I_j.map(\_.assessFitness())$
6: **End for**
7: $result \leftarrow fitnessRDD.collect()$
8: $sortByValue(result)$
9: **If** $result.top = AP$ **then return** top
10: $populationRDD \leftarrow parallelize(PopWithFitness)$
11: **For** each subPopulation $I_j \in populationRDD$ **do**
12: $evolutionRDD \leftarrow I_j.mapPartitions(\_evolution())$
13: $bestIndividualRDD \leftarrow$
$evolutionRDD.map(\_getBest())$
14: **End for**
15: $result \leftarrow bestIndividualRDD.collect()$
16: $sortByValue(result)$
17: **If** $result.top = AP$ **then return** top

FIGURE – Algoritmo Genético Paralelo

Algoritmos

En el algoritmo 1, se pasan tres funciones en el programa del controlador para ejecutar en el clúster : assessF itness (), evolution () y getBest ().
El algoritmo 2 describe el proceso de evolución.

- Toma como entradas la población, el número máximo de generación y el tamaño de población deseado. Las poblaciones evolucionadas en cada trabajador se generan.
- El bucle externo conduce todo el proceso de evolución. En cada iteración del bucle externo, el algoritmo ingresa en un bucle interno que aplica operadores genéticos que incluyen la selección, el cruce, la mutación y el reemplazo.
- Después de abandonar el ciclo interno, el algoritmo muta el mejor individuo. Luego, el algoritmo ingresa al bucle externo para comenzar nuevamente la evolución.
- El proceso de evolución continúa hasta que alcanza el número máximo de generaciones o se ha encontrado la solución. Finalmente, devuelve las poblaciones evolucionadas al Algoritmo 1.

Algoritmos

Input: P: population
max: maximum number of generation
$popsize$: desired population size
Output: evolved populations on every worker

1: $it \leftarrow 1$
2: **While** ($it \leqslant max$ && the ideal solution not found)
3: **For** $popsize/2$ times **do**
4: Parent $P_1 \leftarrow Selection(P)$
5: Parent $P_2 \leftarrow Selection(P)$
6: Children $C_1, C_2 \leftarrow Crossover(Copy(P_1), Copy(P_2))$
7: $Mutate(C_1, C_2)$
8: $ReplaceWorstIndividual(P)$
9: **End for**
10: Mutate the best individual
11: $it \leftarrow it + 1$
12: **End While**
13: Return the evolved populations

FIGURE – Proceso de evolución

EVALUACIÓN PRELIMINAR

Estudiamos el comportamiento de PGAS al resolver un problema de generación de suite de prueba por pares. Realizamos dos categorıas de experimentos : la comparación con el algoritmo genético secuencial (SGA), la comparación con la paralelizada.

Diseño experimental

En los experimentos, implementamos PGAS en Spark usando python y ejecutamos PGAS en un pequeño grupo compuesto por cinco nodos, donde cada nodo tiene un procesador Intel Core i5 750 Quad-Core a 2.66 GHz CPU, 4 GB de RAM. Un nodo es el namenode y los otros cuatro nodos son los nodos de datos que tienen 16 núcleos en total. Cada nodo se ejecuta en Windows 10, Python 3.5 y Spark 1.1.0.
En nuestros experimentos, seleccionamos 14 puntos de referencia sintéticos por pares y 5 puntos de referencia reales. La columna de tuplas representa el número total de pares de cada punto de referencia

Diseño experimental

Hay cinco parámetros que afectan el rendimiento de PGAS : el tamaño de la población, el número de puntos de cruce, el número de puntos de mutación, el número máximo de generaciones y el número de particiones.

- El número de puntos de cruce y el de los puntos de mutación son directamente proporcionales al tamaño del individuo.
- La cantidad de particiones es igual a la cantidad de subpoblaciones generadas por Spark. En general, se establece en 16 según nuestro clúster.
- La configuración de parámetros 1 es la configuración pequeña que se utiliza para los puntos de referencia de tamaño pequeño (S1 S3).
- La configuración de parámetro 2 es la configuración de medio que se utiliza para los puntos de referencia de tamaño medio (S4 S10, SPIN-S, Bugzilla y SPIN-V).
- La configuración de parámetros 3 es la configuración grande que se utiliza para los puntos de referencia de gran tamaño (es decir, S 11 S 14, Apache y GCC).

Parameter name	setting1	setting2	setting3
Population size	4800	8000	12800
Crossover	1-point	3-point	5-point
Mutation	2-point	5-point	5-point
Maximum number of generations	100000	200000	500000

FIGURE – Configuración de parámetros

Diseño experimental

Se hizo una comparacion de tiempos en modo paralelo y serial probando que la forma paralela es mucho mas veloz dando mas eficiencia al algoritmo

Tiempo Paralelo	38.20s	67.28s	122.820s
Tiempo lineal	57.65s	109.46s	264.89s
Tiempo threding(Python)	21.9s	42.5s	102.67s

FIGURE – Resultados

Diagrama de barras

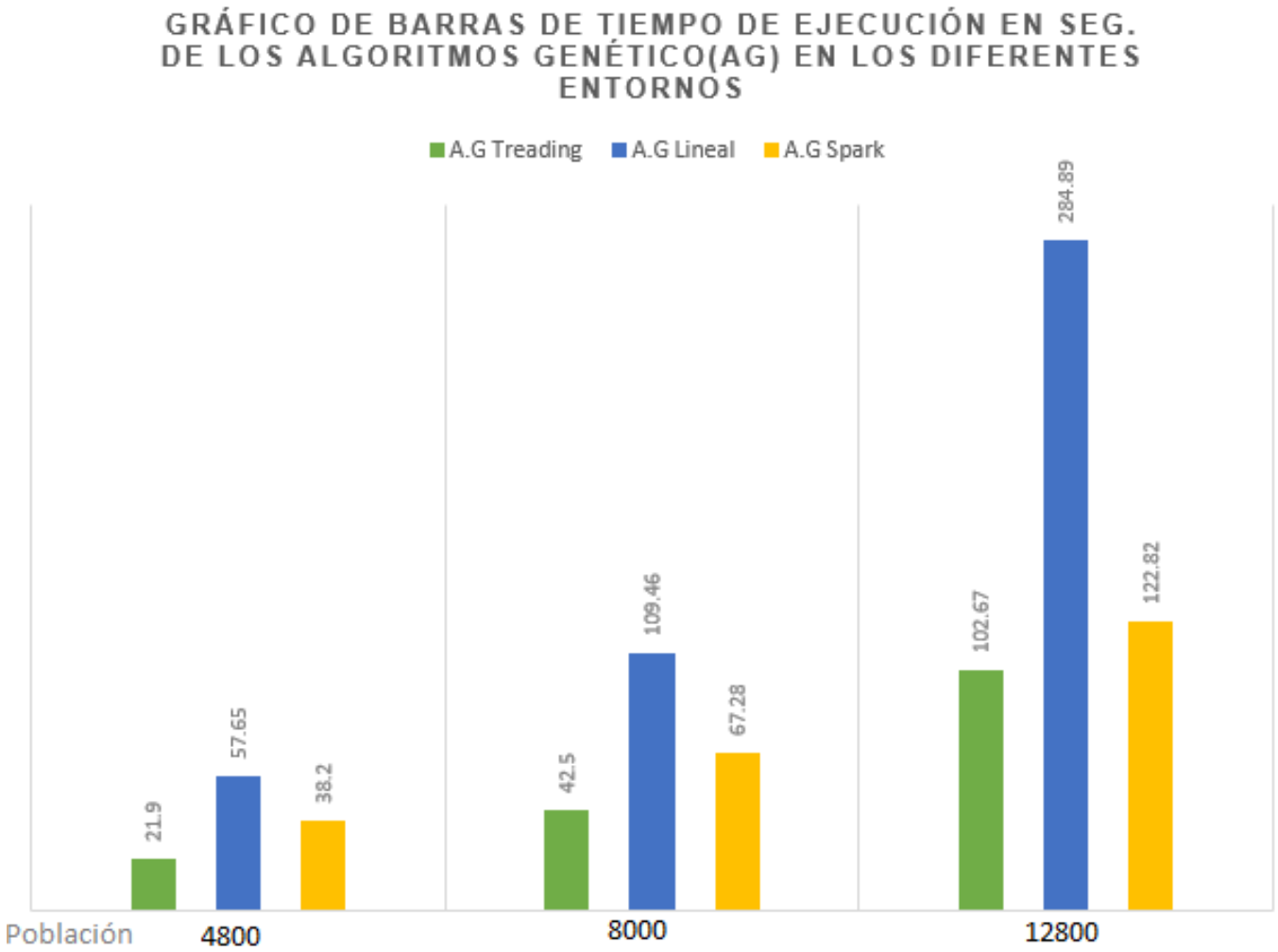

FIGURE – Resultados

Conclusiones

Se propuso un algoritmo genético paralelo basado en threading y se hizo la réplica y un análisis de un algoritmo genético paralelo usando Spark llamado PGAS, para la generación de conjuntos de prueba de pares.
Según nuestros experimento las pruebas realizadas Spark no muestra una diferencia alta en cuanto a eficiencia debido a que la data no es sobrecargada, sin embargo quedo demostrado que cuando la data es considerablemente alta entonces spark respectivamente muestra mejores resultados en cuanto tiempo de ejecución .
Respecto a la eficiencia de spark frente al algoritmo lineal si es de consiederarse.
Observamos que el tiempo de ejecución con los valores mínimos de parámetros tanto en el spark y treading(python) no varian mucho. Realizamos una comparacion donde resulta ser mas eficiente en threding(Python).
Demostramos tambien que la forma paralelizada es mas eficiente que la version serial.
Debido a factores externos a la implementacion del algoritmo los tiempos varian.