Localización de defectos en aplicaciones MapReduce

Jesús Morán[1], Claudio de la Riva[1], Javier Tuya[1], Bibiano Rivas[2]

[1]Departamento de Informática, Universidad de Oviedo, Gijón, España
{moranjesus, claudio, tuya}@uniovi.es
[2]Instituto de Tecnologías y Sistemas de la Información (ITSI), Universidad de Castilla-La Mancha, España
Bibiano.Rivas@uclm.es

Resumen. Los programas que analizan grandes cantidades de datos suelen ejecutarse en entornos distribuidos, tal y como ocurre con las aplicaciones MapReduce. Estos programas se desarrollan independientemente de la infraestructura sobre que la que se ejecutan, ya que un framework gestiona automáticamente la asignación de recursos y gestión de fallos, entre otros. Detectar y localizar defectos en estos programas suele ser una tarea compleja ya que diversas funciones se ejecutan simultáneamente en una infraestructura distribuida, difícil de controlar y que cambia continuamente. En este artículo se describe una técnica que, a partir de un fallo detectado en las pruebas, localiza defectos de diseño analizando dinámicamente los parámetros que lo causan.

Palabras clave: Pruebas del software, Localización de defectos, MapReduce, Big Data Engineering

1 Introducción

Entre las nuevas tecnologías de análisis masivo de datos denominadas Big Data, se destaca el modelo de procesamiento *MapReduce* [1]. Estos programas son ejecutados por un framework (ej. Hadoop) que automáticamente realiza el despliegue en una infraestructura distribuida y se encarga de que el programa re-ejecute parte de los datos en caso de fallos de infraestructura, entre otros. De esta forma los programas se pueden desarrollar independientemente de la infraestructura sobre la que se ejecutarán. A cambio, se tiene que considerar cómo ésta puede afectarle a la funcionalidad.

En un trabajo anterior [2] se han identificado defectos funcionales que dependen de cómo la configuración de la infraestructura afecta a la ejecución del programa e influye en su resultado. Estos defectos suelen enmascararse durante las pruebas si éstas se ejecutan sobre una configuración que no considera diferentes situaciones que pueden ocurrir en producción, como el paralelismo o posibles fallos en la infraestructura. En otro trabajo anterior [3], se ha diseñado una técnica de ejecución de pruebas que detecta automáticamente estos defectos en un entorno de desarrollo. Tras detectarlos, es difícil localizar su causa raíz debido a las interacciones de varias funciones ejecutadas en paralelo. Este artículo extiende los trabajos anteriores con una técnica para localizar los defectos ejecutando dinámicamente el programa en diferentes configuraciones.

2 MapReduce

Los programas *MapReduce* siguen el principio “divide y vencerás” con dos funcionalidades: Mapper divide un problema en varios subproblemas, y Reducer resuelve cada uno de ellos en paralelo. Dado que suelen utilizar intensivamente la red, se pueden realizar optimizaciones implementando una funcionalidad llamada Combiner. Esta tarea precalcula el resultado de los subproblemas con los datos disponibles localmente en las tareas Mapper. Suponer a modo de ejemplo un programa que calcula la temperatura media por cada año. Esta aplicación tiene tantos subproblemas como años, de forma que: (1) Mapper recibe un subconjunto de <año, temperatura> y los agrupa por año, y (2) se calcula la temperatura media por año mediante la tarea Combiner con la información local, y Reducer con la información global.

Dependiendo de la configuración de infraestructura estos programas se pueden ejecutar de diferente forma aun recibiendo la misma entrada. En la figura 1 se describe la ejecución del programa en tres configuraciones diferentes recibiendo ambas las temperaturas 4º, 2º y 3º, en 1999. La primera configuración es la habitual de un entorno de pruebas que se ejecuta sin paralelismo y produce la salida esperada. Las otras dos configuraciones ejecutan el programa con paralelismo. La última produce un fallo porque este programa se debe diseñar sin tarea Combiner, ya que para calcular la temperatura media se necesitan todas las temperaturas del año y no unas pocas.

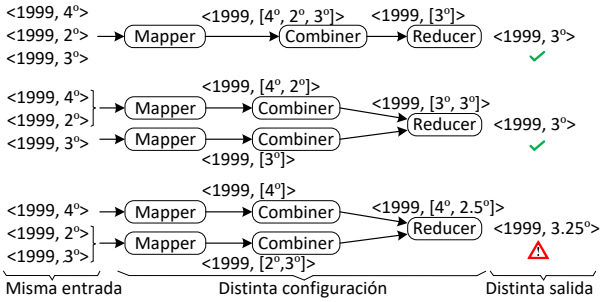

Fig. 1. Ejecución de pruebas para un programa que calcula la temperatura media por año

3 Localización de Defectos

Tras detectar un defecto, es difícil localizarlo observando estáticamente la configuración de infraestructura y el fallo producido, ya que depende de la interacción paralela de las diferentes tareas *MapReduce*, así como de los datos de entrada y la complejidad de la aplicación. La técnica de localización que se propone, parte de la configuración que produce el fallo (**Caracterización**), para generar dinámicamente nuevas configuraciones de infraestructura que comparten todos sus parámetros excepto uno que se modifica para observar de forma controlada si afecta o no al resultado (**Generación**). Finalmente, el defecto se localiza a través del parámetro que al modificarlo pasa de producir una salida a causar otra diferente (**Análisis**). En el resto de la sección se describe cada uno de los anteriores pasos, y su aplicación en el programa de la sección 2.

Caracterización de la configuración que provocó un fallo en la fase de pruebas. Para ello se utilizan clases de equivalencia que, combinándolas, representan las configuraciones de infraestructura típicas que pueden tener los programas *MapReduce*:

- Mapper:
  1. Número de Mappers: una o varias.
  2. Orden de ejecución: datos ejecutados en el mismo orden que están en la entrada, o en diferente orden.
  3. Distribución de datos entre Mappers: datos distribuidos equitativamente o no.
- Combiner:
  4. Número de Combiners: cero, una o varias.
  5. Distribución de datos entre Combiners: datos distribuidos equitativamente o no.
  6. Datos directos a Reducer sin pasar por Combiner: cero o más de cero.
  7. Ejecuciones iterativas de Combiner: una o varias.
- Reducer:
  8. Número de Reducers: una o varias.

En el centro de la figura 2 se tiene la configuración que produjo un fallo en las pruebas. Esta se caracteriza con las clases de equivalencia, obteniendo que tiene varias tareas Mapper y que ejecuta los datos en el mismo orden que están en la entrada, entre otras.

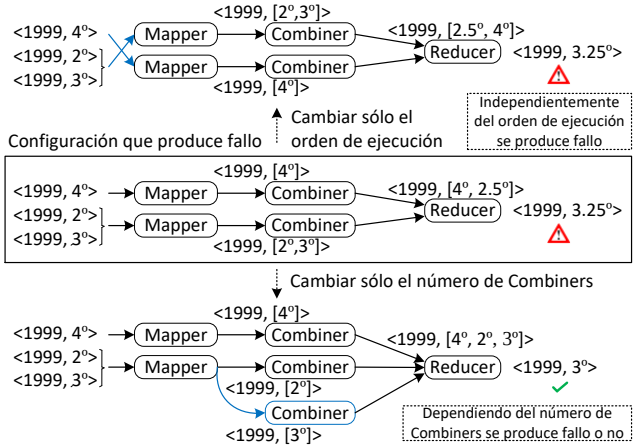

Fig. 2. Localización de defecto en un programa que calcula la temperatura media por año

Generación y ejecución de nuevas configuraciones manteniendo todas las clases de equivalencia excepto una que se modifica de forma controlada para observar qué efecto tiene en la salida. Por ejemplo, la configuración de la parte superior de la figura 2 modifica únicamente el orden de ejecución de las tareas Mapper, y la configuración de la parte inferior, el número de Combiners. Independientemente del orden de ejecución el resultado es el mismo, por tanto, parece que no es la causa del fallo. En cambio, al cambiar el número de Combiners se produce la salida esperada, lo cual indica que dependiendo del número de Combiners unas veces produce la salida esperada y otras no. Para obtener una mejor localización se generan y ejecutan varias configuraciones.

Análisis de la ejecución de las pruebas. Para este fin se consideran las clases de equivalencia de las ejecuciones así como su resultado, utilizando la medida Ochiai por ser de las que mejores localizando defectos [4]. Este análisis aísla y prioriza las clases de equivalencia que con un valor enmascaran el defecto y con otro valor causan fallo. Para el programa que calcula la temperatura media se han ejecutado 13 configuraciones, obteniendo que las clases de equivalencia potencialmente más problemáticas son: una única Combiner, cero datos que van directamente a Reducer sin pasar por Combiner, datos equitativamente distribuidos en Combiner, y una ejecución iterativa de Combiner. Efectivamente el programa no admite esa tarea Combiner.

4 Trabajo Futuro

Este artículo describe una técnica de localización de defectos funcionales para aplicaciones *MapReduce* que se ha integrado y automatizado en un motor de ejecución de pruebas. Partiendo de una configuración que produce fallo, se generan y ejecutan dinámicamente nuevas configuraciones para analizar qué parámetros de la configuración causan el fallo. Como trabajo futuro se está experimentando con varias medidas de localización en distintos programas. También se está integrando la técnica de localización con pruebas realizadas automáticamente en el entorno producción (on-line).

Agradecimientos

A la Dra. Antonia Bertolino (ISTI-CNR, Pisa Italia) por sus contribuciones en la línea de investigación. Este trabajo ha sido realizado bajo los proyectos de investigación TIN2016-76956-C3-1-R (TESTEAMOS), TIN2015-63502-C3-1-R (SEQUOIA) y TIN2013-46928-C3-1-R (PERTEST), financiados por el Ministerio de Economía y Competitividad, y fondos FEDER. También ha sido realizado bajo el proyecto GRUPIN14-007, financiado por el Principado de Asturias y fondos FEDER.

Referencias

1. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Proc. OSDI - Symp. Oper. Syst. Des. Implement. 137–149 (2004).
2. Moran, J., Riva, C. de la, Tuya, J.: MRTree: Functional Testing Based on MapReduce's Execution Behaviour. In: 2014 International Conference on Future Internet of Things and Cloud. pp. 379–384. IEEE (2014).
3. Morán, J., Rivas, B., Riva, C. De, Tuya, J., Caballero, I., Serrano, M.: Configuration / Infrastructure-aware testing of MapReduce programs. Adv. Sci. Technol. Eng. Syst. J. 2, 90–96 (2017).
4. Abreu, R., Zoeteweij, P., Van Gemund, A.: An Evaluation of Similarity Coefficients for Software Fault Localization. In: 2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06). pp. 39–46. IEEE (2006).