Automatización de la localización de defectos en el diseño de aplicaciones MapReduce

Jesús Morán, Claudio de la Riva, Javier Tuya

Departamento de Informática, Universidad de Oviedo, Gijón, España
{moranjesus, claudio, tuya}@uniovi.es

Resumen. Los programas MapReduce analizan grandes cantidades de datos sobre una infraestructura distribuida. En cambio, estos programas pueden desarrollarse independientemente de la infraestructura ya que un framework gestiona automáticamente la asignación de recursos y la gestión de fallos. Una vez que se detecta un defecto, suele ser complicado localizar su causa raíz ya que diversas funciones se ejecutan simultáneamente en una infraestructura distribuida que cambia continuamente y que es difícil tanto de controlar como depurar. En este artículo se describe una técnica que, a partir de un caso de prueba que produce fallo, localiza su causa raíz analizando dinámicamente las características del diseño que se cubren cuando se produce fallo y aquellas que no.

Palabras clave: Pruebas del software, Localización de defectos, MapReduce

1 Introducción

MapReduce [1] es uno de los modelos de programación más empleados para análisis masivo de datos. La ejecución de los programas es gestionada por un framework (ej. Hadoop) que automáticamente realiza el despliegue en una infraestructura distribuida y re-ejecuta parte de los datos en caso de fallos de infraestructura, entre otros. A cambio, en el diseño se tiene que considerar cómo ésta puede afectarle a la funcionalidad.

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 se puede diseñar con 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.

En un trabajo anterior [2] se identifican defectos funcionales en el diseño de los programas *MapReduce*, por los cuales la configuración de la infraestructura afecta al 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. La figura 1 describe la ejecución del programa anterior en dos configuraciones diferentes recibiendo ambas las temperaturas 4º, 2º y 3º, en 1999. La primer configuración es la habitual del entorno de pruebas, se ejecuta sin paralelismo y produce la salida esperada. En cambio, la otra configuración ejecuta paralelismo y 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.

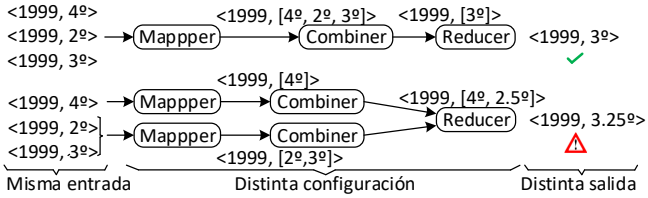

Fig. 1. Ejecución de pruebas para un programa que calcula la temperatura media por año

Los defectos como el anterior, causados por un incorrecto diseño del programa, se pueden detectar de forma automática mediante la herramienta MRTest [3]. Tras detectarlos, es difícil localizar su causa raíz debido a las interacciones de varias funciones ejecutadas en paralelo. En otro trabajo [4] se comenzó a diseñar una técnica para localizar estos defectos de forma automática. Este artículo extiende el trabajo anterior integrando las pruebas con la localización de los defectos mediante la generación dinámica de diferentes configuraciones y el análisis de las posibles causas raíz.

.

2 Técnica de Localización de Defectos

Este artículo propone localizar automáticamente los defectos en el diseño de las aplicaciones *MapReduce* mediante la técnica representada en la figura 2. A partir de un caso de prueba (datos y configuración) con resultado de fallo, se generan y ejecutan nuevas configuraciones para obtener las causas más sospechosas del fallo monitorizando y analizando las características que tienen en común aquellas configuraciones que provocan fallo en el programa y las que no. En el resto de sección se describe brevemente las diferentes partes de la técnica de localización.

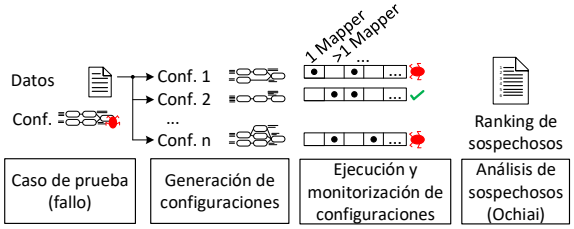

Fig. 2. Técnica de localización de defectos funcionales en MapReduce

Caso de prueba: la técnica de localización recibe como entrada un caso de prueba que produce un fallo a causa de un diseño incorrecto del programa. Estos casos de

prueba se pueden generar manualmente apoyándose en la herramienta MRTest [3] o automáticamente a partir de los datos que existen en producción [5].

Generación de configuraciones: la técnica genera aleatoriamente varias configuraciones con el objetivo de analizar qué características del diseño son las que causan el fallo. Para ello, se considera que las configuraciones pueden tener distinto número de Mappers/Combiners/Reducers, cada Mapper/Combiner/Reducer puede recibir distinto número de datos, las Mapper se pueden ejecutar en distinto orden, y las Combiner se pueden ejecutar de forma iterativa un número distinto de veces.

Ejecución y monitorización de configuraciones: las configuraciones generadas en el paso anterior se ejecutan junto con los datos del caso de prueba que se recibe como entrada, monitorizando por cada ejecución: (1) si el defecto se enmascara o no, y (2) las características que cubre cada configuración enumeradas en el párrafo anterior, entre otros, el número de Mappers y el número de datos que ejecuta cada Mapper. Para la ejecución se utiliza la herramienta MRTest [3] que de forma automática determina si se produce fallo o no al comparar las salidas de varias configuraciones.

Análisis de sospechosos: a partir de las características de las configuraciones que producen fallo y las que no, se realiza un análisis para determinar cuáles son las más sospechosas de causar el fallo. Para este fin se emplea una métrica comúnmente empleada en la localización de defectos, Ochiai [6]. Esta métrica no sólo tiene en cuenta las características que ocurren cuando hay fallo, sino también aquellas que enmascaran el defecto. Finalmente, se obtiene un ranking ordenado con las características más sospechosas de causar el fallo.

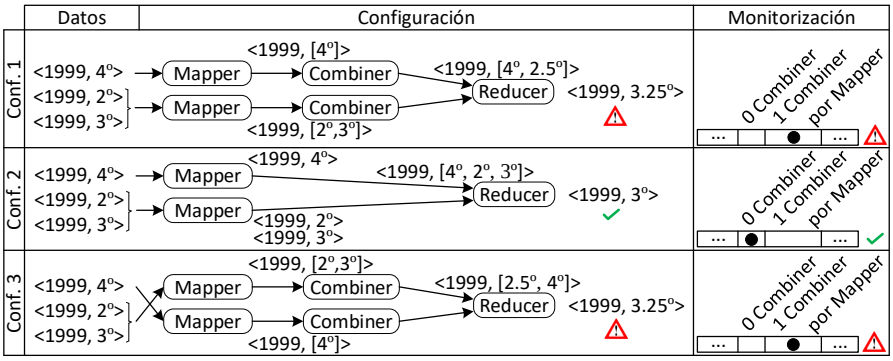

Fig. 3. Localización de defecto en un programa que calcula la temperatura media por año

En la figura 3 se muestra una parte de la técnica de localización aplicada al programa presentado en la sección 1 que calcula la temperatura media por año. Se parte de un caso de prueba que produce fallo, por ejemplo el año 1999 con las temperaturas 4º, 2º y 3º, tal y como se representa en la figura 1. A continuación, se generan y ejecutan automáticamente varias configuraciones (en la figura sólo se muestran tres), monitorizando que dos configuraciones ejecutan Combiner y producen fallo (Conf.1 y Conf. 3), mientras que otra configuración no ejecuta Combiner y no produce fallo

(Conf. 2). Finalmente se realiza un análisis con la métrica Ochiai en el que se concluye que la causa más probable del fallo es la ejecución de Combiner ya que (1) aquellas configuraciones que ejecutan Combiner suelen causar fallo, y (2) las que no ejecutan Combiner no suelen causar fallo. En el caso anterior, Ochiai asigna al más sospechoso (ejecución de Combiner) el valor 0.7, mientras que para el resto le asignan valores más bajos, entre 0 y 0.55. La técnica de localización localiza correctamente la causa raíz del fallo ya que el programa no admite esa tarea Combiner.

3 Trabajo Futuro

En este artículo se describe una técnica para localizar defectos funcionales en el diseño de aplicaciones *MapReduce*. Partiendo de un caso de prueba que produce fallo, se generan y ejecutan dinámicamente nuevas configuraciones para analizar qué características de la configuración causan el fallo. Como trabajo futuro se pretende elaborar una técnica que genere las configuraciones más relevantes para localizar los defectos, así como experimentar con varias métricas de localización en distintos programas.

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 y TIN2013-46928-C3-1-R, 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. Morán, J., De La Riva, C., Tuya, J., Rivas, B.: Localización de defectos en aplicaciones MapReduce. In: Jornadas de Ingeniería del Software y Bases de Datos (2017).
5. Moran, J., Bertolino, A., de la Riva, C., Tuya, J.: Towards Ex Vivo Testing of MapReduce Applications. In: 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS). pp. 73–80. IEEE (2017).
6. 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).