# 분산 어플리케이션 로그 분석을 통한 설정 오류 문제 원인 식별

양지혁° 탁병철
경북대학교 컴퓨터학부
flash3470@gmail.com, bctak@knu.ac.kr

# Identifying Root Cause of Misconfiguration by Analyzing Distributed Application Logs

Jihyeok Yang°, Byungchul Tak
School of Computer Science and Engineering, Kyungpook National University

**요 약**

오늘날 다수의 서버 노드로 구성되는 빅데이터 시스템을 사용하는 기업이 많아졌다. 시스템 관리자는 노드가 많아짐에 따라 관리해야 할 설정 파일과 설정 값이 많아져 설정 오류를 일으킬 확률과 해결 시간이 증가하였다. 설정 오류가 일어나면 시스템의 동작에 이상이 나타나며, 이는 곧 기업의 수익과 직결된 일이기에 빠르게 해결 해야한다. 본 연구에서는 설정 오류를 빠르게 해결하기 위하여 한 가지 설정 값에 잘못된 값을 삽입하여 동작 시킨 시스템의 로그와 정상 시스템 로그의 로그 패턴을 몇가지 로그 특성을 나타내는 지표 값인 메트릭(Metric)으로 분석하는 작업을 모든 설정 값에 대해 수행하고, 각 설정 값에 대한 메트릭 값의 테이블을 만든다. 이 테이블을 바탕으로 문제가 생긴 시스템의 로그의 패턴을 테이블을 생성할 때 사용했던 메트릭으로 분석하여 나온 값을 메트릭 테이블과 비교해 문제가 될 후보 설정 값 명단을 만들고, 이를 가능성이 높은 순으로 순위를 매겨 설정 오류를 일으킨 설정 값을 진단하는 기법을 제안한다.

## 1. 서 론

오늘날 많은 서비스 제공업체들은 빅데이터 시스템에 관심을 가지고 도입을 하기 시작했다. 일반적인 빅데이터 시스템은 보통 수십 테라바이트에서 수 페타바이트의 데이터를 분석하게 되는데, 이러한 큰 데이터를 처리하기 위해서는 많은 양의 시스템 자원을 필요로 한다. 이 때문에 빅데이터 시스템은 대게 분산 처리를 할 수 있는 시스템으로 구성되도록 권장되며 이는 다수의 서버 노드를 필요로 한다.

다수의 서버 노드로 구성된 시스템은 단일 서버 시스템보다 복잡한 구조를 가지게 되고, 이러한 구조의 빅데이터 시스템을 구성하기 위해 다양한 설정 파일과 그 설정 파일에 속한 다양한 설정 값이 존재한다. 시스템 관리자가 자신이 원하는 동작을 하는 시스템을 구성하기 위해서는 설정 파일과 값에 대해 이해한 후 수정해야 한다. 이 과정에서 시스템 관리자가 설정 오류를 일으킬 수 있다.

여기서 설정 오류(misconfiguration)는 설정 파일의 설정 값을 잘못 설정하여 시스템의 동작에 문제가 생기는 것을 의미한다. 이는 설정 해야할 설정 값의 개수가 많을수록 더 빈번히 일어난다. 예를 들어, 하둡 시스템 구성 시 최소 4개의 설정 파일에 대하여 설정을 해줘야 하며, 각 설정 파일의 설정 값의 개수는 200~300개 다. 즉, 천 개 정도의 설정 값 중 몇 개의 설정 값을 수정하여 설정 오류를 일으키는 것은 어렵지 않은 일이다[1].

이러한 설정 오류는 시스템 상의 다양한 문제를 일으킨다. 일반적으로 설정 오류가 발생하면 시스템이 정상 동작을 하지 않거나 프로그램 수행 시간이 길어지는 것과 같은 문제를 발생시킨다. 이러한 빅데이터 시스템의 장애들은 시스템을 구성한 회사의 수익과 밀접한 연관이 되는 요소이기 때문에 빨리 설정 오류의 원인이 되는 설정 값을 찾고 고치는 능력은 중요한 문제이다.

하지만 시스템 관리자가 설정 오류를 해결하는 것은 매우 어려운일 이다. 일반적으로 시스템 관리자는 설정 오류의 원인을 알기 위해 시스템에서 나오는 로그에 의존할 수밖에 없다. 로그를 직접보고 수동으로 수많은 로그를 분석하여 해결하거나 백업 해놓은 설정 파일과 자신의 설정 파일의 값들을 일일이 비교하여 설정 오류를 해결한다. 이러한 해결법들은 항상 많은 시간을 소요하기 때문에 설정 오류를 해결하는 방법으로 적합하지 않다.

그리고 설정 오류를 해결하기 위해 시도된 기존의 연구는 프로그램의 소스코드를 분석하는 white-box 접근 기법[2,3], 각 설정 값의 상관 관계를 분석하는 기법[4]과 설정 파일을 다른 컴퓨터들의 설정 파일과 비교하여 설정오류를 찾는 기법 등이 있다[5, 6]. 하지만 이들은 소스코드 내부를 알아야 하거나[2,3] 수동적인 상관 관계의 분석[4] 혹은 각 설정 파일에 대한 수동적인 라벨링이 필요하다는 단점을 가진다[5]. 그리고 [5]의 수동적인 분석이라는 단점을 보완한[6]의 경우 오직 20개의 케이스에 대해서만 다룬다.

본 연구는 시스템 관리자가 빠르게 설정 오류를 해결할 수 있도록 정상의 설정 파일에서 하나의 설정 값이 잘못된 값을 가지도록 오류를 삽입(fault injection)하여 시스템을 동작 시켜서 로그를 만들어 내는 과정을 설정 파일 내 모든 설정 값에 대해 진행한다. 그리고 하나의 오류가 주입된 로그와 정상 시스템 로그에서 나타난 각 로그 라인들의 로그 패턴 집합의 유사도와 같이 각 로그의 특성을 나타내는 지표 값인 메트릭(Metric)에 대한 테이블을 만든다. 그리고 이를 바탕으로 실제 설정 값을

바꾼 후 시스템에 문제가 생겼을 때 나타나는 로그 패턴을 분석하여 후보 설정 값 명단을 생성하고, 모든 메트릭 값에 있어서 두 유사도가 가장 비슷한 설정 값 순으로 나열하여 설정 오류로 인한 시스템의 장애의 원인이 되는 설정 값을 추정하는 기법을 제안하고 이를 Spark시스템을 구성하여 적용해보았다.

## 2. 문제가 되는 설정 값 진단 과정

본 연구에서 문제가 되는 설정 값을 진단하는 방법을 2.1~2.3장의 과정을 순차적으로 진행하면서 보여 준다.

### 2.1 메트릭(metric) 테이블 생성

먼저 정상적인 설정 파일에 하나의 설정 값을 잘못된 값으로 설정한 후, 해당 시스템을 동작 시켜서 나온 로그와 올바른 설정 파일을 가지고 시스템을 동작 시켜서 나온 로그에 대해 하나의 메트릭 값을 구한다. 그리고 이 과정을 시스템의 여러 상황에 이 과정을 반복하여 그 설정 값에 대한 여러 개의 메트릭 값을 모은다. 그리고 이 여러 개의 메트릭 값을 바탕으로 그 잘못된 설정 값에 대한 메트릭 값의 평균과 표준 편차를 하나의 row로 기록하고, 앞의 모든 과정에 대해서 모든 설정 값들을 대상으로 반복하여, 한 메트릭에 대한 테이블을 만든다. 구성된 테이블은 아래의 Figure 1의 형태를 가진다. 이 논문에서는 로그 패턴의 집합에 대한 유사도를 나타내는 자카드 거리와 나타난 각 로그 패턴들의 개수의 유사도를 나타내는 코사인 거리를 메트릭으로 사용하였고 이 두 메트릭에 대한 테이블을 각각 구성하였다.

| Jaccard Distance | 평균 | 표준편차 |
|---|---|---|
| 설정 값1 | 0.6 | 0.1 |
| 설정 값2 | 0.4 | 0.05 |
| … | | |
| 설정 값n | | |

Figure 1. 메트릭 테이블 구성 예시

### 2.2 후보 설정 값 명단 생성

2.1장에서 생성된 메트릭 테이블을 바탕으로 후보 설정 값 명단을 생성해야한다. 후보 설정 값 명단은 설정 오류를 찾고 싶은 로그 $X$에 대한 메트릭 $K$의 분석 값 $X_K$가 $K$ 메트릭 테이블에서 설정 값 $C$의 분석 값 $C_K$의 평균 $M(C_K)$, 표준편차 $S(C_K)$에 대해 아래의 Figure 2 수식의 조건을 적어도 하나의 메트릭 테이블에 대해서 만족하는 모든 $C$ 로 구성된다. 예를 들면, $X_K$에 대해 설정 값1은 Figure 2 수식을 만족하기 때문에 $X$의 후보 설정 값 명단에 포함된다.

$$M(C_K) - 1.96 * S(C_K) \le X_K \le M(C_K) + 1.96 * S(C_K)$$

Figure 2. 신뢰 구간을 이용한 후보 설정 값 판단 수식

### 2.3 후보 설정 값 리스트 정렬

이 2.3장은 2.2장에 의해 생성된 설정 오류에 대한 후보 설정 값 리스트에서 가장 가능성이 높은 후보 설정 값 순으로 정렬하는 것은 메트릭의 총 개수$CNT(K)$, 설정 값 $C$가 Figure 2 수식을 만족하는 메트릭 $K$인 $K_C$의 개수 $CNT(K_C)$, 그리고 $X_K$와 $C_K$의 차의 제곱을 나타내는 $SE(C)$로 구성된 값 P의 내림차순으로 정렬된다. P를 구하는 수식은 아래 Figure 3 과 같다.

$$\mathrm{P} = \frac{CNT(K_C)}{CNT(K)} \times (1 - Average\ of\ SE(C))$$

Figure 3. 후보 설정 값 명단 정렬 기준 값 P

## 3. 실험 결과

### 3.1 실험 구성

본 연구를 위해 Apache Spark 시스템을 구성하여 로그를 수집하였다. 로그 수집에 사용된 Spark 시스템은 하나의 마스터 노드와 두 개의 슬레이브 노드로 구성된다. 그리고 각 노드는 하나의 가상머신 내에 각각의 도커 컨테이너로 구성 했으며 가상 머신은 하나의 호스트PC에 여러 개가 동작한다. 그리고 호스트 PC에서 가상 머신이 만들어낸 로그를 Metric Analysis 컨테이너들에서 분석한 값을 DB서버 Container의 DB에 넣는다. 그리고 이를 바탕으로 Metric Table Creator 컨테이너에서 메트릭 테이블을 생성한 후 Misconfig Diagnose 컨테이너에서 설정 오류 탐지가 이루어진다. 이러한 전체 시스템의 구조도를 Figure 4로 나타냈다.

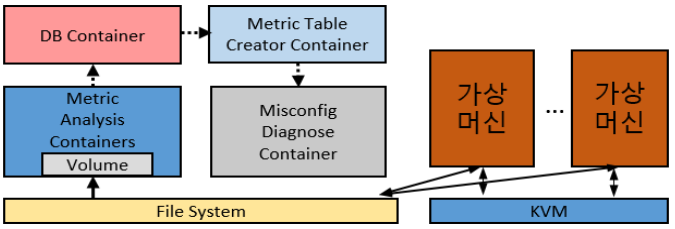


Figure 4. 전체 시스템 구성도

메트릭 테이블을 구성하기 위해서 두 가지 설정 파일 중에서 하나의 설정 값을 잘못된 값으로 설정한 후, 세 가지 Spark app에 대하여 랜덤하게 실행하여 5~10개에 해당하는 랜덤한 횟수의 로그를 수집하는 과정을 모든 설정 값에 대해 실행 하였다. 그리고 본 연구 기법의 정확도를 테스트 하기 위해 각 설정 값마다 그 설정 값을 잘못된 값으로 설정한 후, 하나의 Spark app를 실행해서 나온 로그의 메트릭 값 들로 설정 오류 진단을 하였을 때 그 설정 값이 후보 설정 값 명단에 포함되는지, 몇 위에 존재하는지에 대해 판단한다. 자세한 실험 정보에 대한 것은 Table 1에 정리하였다.

| 타겟 설정 파일 | yarn-site.xml, hdfs-site.xml |
|---|---|
| 총 설정 값 개수 | 504 |
| 로그 수집 시 사용한 Spark app | wordcount, logistic regression, pi |
| 설정 오류 진단 테스트 Spark app | pagerank |

Table 1. 실험 정보 표

### 3.2 실험 결과 및 분석

#### 3.2.1 설정 오류 진단 결과

3,1에서 구성된 실험의 결과로 설정 오류 진단 결과로 후보 설정 값 명단을 만들어 1등인 경우, 각 순위(2~20, 21~100, …, 401~500)내에 속하는 경우, 못 찾는 경우에 대하여 yarn-site.xml에서 분포, hdfs-site.xml에서의 분포, 그리고 두가지 설정 파일 전체에 대한 분포를 Table 2 표에 나타내었다. 그리고 Rank 1인 설정 값에 대해 좀 더 자세히 조사를 해 그 중 일부 설정 값에 대해 Table 3에 나타냈다. Table 3에서 각각의 설정 값의 타입은 랭킹은 제각기 달라

상관 없어보였으며, 메트릭 테이블에서 표준 편차가 매우 적으며 메트릭 값들이 작음을 알 수 있다. 이는 그 설정 값에 잘못된 값을 넣으면 사용한 4가지 스파크 앱 모두 실행 시켰을 때, 로그 패턴이 일정하게 정상 시스템 로그 패턴과 다르다는 것을 보여준다.

| Rank | yarn-site.xml | hdfs-site.xml | **총 개수** |
|---|---|---|---|
| **1** | **1** | **13** | **14** |
| **2~20** | **10** | **44** | **54** |
| 21~100 | 21 | 70 | 91 |
| 101~200 | 25 | 75 | 100 |
| 201~300 | 55 | 46 | 101 |
| 301~400 | 74 | 28 | 102 |
| 401~500 | 22 | 19 | 41 |
| **Not Found** | **0** | **1** | **1** |
| Total | 208 | 296 | 504 |

Table 2. 실험 결과 분포표

| | Value Type | cosine distance | | jaccard distance | |
|---|---|---|---|---|---|
| | | mean | stddev | mean | stddev |
| dfs.bytes-per-checksum | Numeric | 0.250950 | 0.000110 | 0.605711 | 0.001368 |
| dfs.data.transfer.protection | List | 0.250423 | 0.000111 | 0.509051 | 0.001152 |
| dfs.namenode.lifeline.rpc-address | IP address | 0.250089 | 0.000112 | 0.449052 | 0.001002 |
| dfs.namenode.shared.edits.dir | Directory | 0.250052 | 0.000110 | 0.367036 | 0.000714 |
| dfs.namenode.plugins | List | 0.250108 | 0.000112 | 0.475591 | 0.001059 |

Table 3. Rank 1 설정 값 분석 표

### 3.2.2. 진단 결과 분석

3.2.1을 보면 약 88%정도의 상당한 설정 값 오류에 대해 진단 하지 못했다. 진단 결과가 왜 이렇게 나왔는지 메트릭 테이블을 분석해본 결과 원인을 다음 두가지로 생각하였다. 첫 번째 원인은 메트릭 테이블에서 메트릭 평균 칼럼에 대해 1에 가까운 설정 값이 많았다는 것이다. 평균 값이 1인 설정 값에 대해 그 설정 값에 잘못된 값을 넣어 실행한 시스템 로그와 정상 시스템 로그에서 나타난 각 로그 라인들의 로그 패턴 차이가 크게 없음을 뜻한다. 잘못된 값을 넣었을 때 로그 패턴의 차이가 크게 없다는 것은 프로그램의 흐름에 그 설정 값이 크게 영향을 미치는 부분이 없었다는 것이다. 그리고 두 번째 원인으로는 메트릭 테이블에서 표준편차 값이 큰 값들이 존재하는 것이다. 메트릭 테이블에서 한 설정 값의 표준편차 값이 크다는 것은 그 설정 값이 잘못 되었을 때 로그 패턴의 변화가 각 프로그램별로 일정한 값을 가지지 못한다는 것을 뜻한다. 이는 주로 일부 프로그램이 설정 값이 영향을 미치기 전에 종료 되면서 발생된다. 이 두 원인이 얼마나 실험에 영향을 미치는지 파악하기 위해 메트릭 값이 0.99이상인 설정 값 개수와 표준편차가 0.1 이상인 설정 값 개수를 각 메트릭 별로 Table 4에 정리하였고, 두 메트릭 값이 동시에 0.99이상이거나 표준편차가 0.1이상인 설정 값의 개수에 대해 Table 5에 정리 하였다. 그리고 이를 통해 두 원인이 나타나는 설정 값 개수와 설정 오류 진단을 잘 하지 못한 원인 설정 값 개수랑 비슷하다는 것을 알 수 있다.

| | Jaccard Distance Table | Cosine Distance Table |
|---|---|---|
| 평균 값>0.99 | 393 | 436 |
| 표준편차>0.1 | 31 | 20 |

Table 4. 메트릭 테이블 별 이상 설정 값 개수 표

| | 두 테이블 동시에 나타나는 이상 값 개수 |
|---|---|
| 평균 값>0.99 | 392 |
| 표준편차>0.1 | 19 |

Table 5. 두 메트릭 테이블에서 동시에 이상 값을 가지는 설정 값 개수 표

## 4. 결론 및 향후 과제

3장의 실험에서 두가지 메트릭을 사용하여 설정 오류를 탐지 하였을 때 전체 설정 값의 상위 20위에 랭크하는 경우가 68개로 전체 중 12%에 불과했다. 하지만 본 연구와 같이 기법에 수동적 분석 부분이 없는 [6]의 실험에서 설정 오류 진단 케이스 개수인 20개에 비하면 많은 개수의 케이스에 대해 잘 진단되었지만 진단 정확도가 다소 떨어진다. 진단 정확도를 높이기 위해 향후 로그 패턴의 유사도를 측정하는 메트릭 이외의 새로운 로그 값의 변화를 나타낼 수 있는 메트릭을 추가 할 것 이다. 그리고 3.2.2장에서의 두 가지 원인에 대해서 더 정밀한 분석을 해서 이를 바탕으로 설정 오류 진단의 정확도를 더 높이는 방법을 찾는 것이 가장 중요한 과제이다. 그리고 이를 오픈스택과 같은 다른 분산 시스템에도 적용을 해보는 것도 향후 진행 과제 중 하나다.

## 6. 참고 문헌


[1] https://hadoop.apache.org/

[2] M. Attariyan, M. Chow, and J. Flinn. *X-ray: Diagnosing Performance Misconfigurations in Production Software.* In Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI'12), October 2012

[3] A. Rabkin and R. Katz. *Precomputing Possible Configuration Error Diagnoses.* In Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ICSE'11), May 2011.

[4]Zhang, J., Renganarayana, L., Zhang, X., Ge, N., Bala, V., Xu, T., AND Zhou, Y. *EnCore: Exploiting System Environment and Correlation Information for Misconfiguration Detection.* In Proceedings of the 19th International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS'14) (Salt Lake City, UT, USA, Mar. 2014).

[5] Y.-M. Wang, C. Verbowski, J. Dunagan, Y. Chen, H. Wang, C. Yuan, and Z. Zhang. *STRIDER: A Black-box, Statebased Approach to Change and Configuration Management and Support.* In Proceedings of the 17th Large Installation Systems Admistration Conference (LISA'03), October 2003.

[6] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang. *Automatic Misconfiguration Troubleshooting with PeerPressure.* In Proceedings of the 6th USENIX Conference on Operating Systems Design and Implementation (OSDI'04), December 2004.