KIPS Trans. Comp. and Comm. Sys.
Vol.8, No.2 pp.35~40 pISSN: 2287-5891

https://doi.org/10.3745/KTCCS.2019.8.2.35

Distributed Processing System Design and Implementation for Feature Extraction from Large-Scale Malicious Code

Hyunjong Lee† · Seongyul Euh†† · Doosung Hwang†††

ABSTRACT

Traditional Malware Detection is susceptible for detecting malware which is modified by polymorphism or obfuscation technology. By learning patterns that are embedded in malware code, machine learning algorithms can detect similar behaviors and replace the current detection methods. Data must collected continuously in order to learn malicious code patterns that change over time. However, the process of storing and processing a large amount of malware files is accompanied by high space and time complexity. In this paper, an HDFS-based distributed processing system is designed to reduce space complexity and accelerate feature extraction time. Using a distributed processing system, we extract two API features based on filtering basis, 2-gram feature and APICFG feature and the generalization performance of ensemble learning models is compared. In experiments, the time complexity of the feature extraction was improved about 3.75 times faster than the processing time of a single computer, and the space complexity was about 5 times more efficient. The 2-gram feature was the best when comparing the classification performance by feature, but the learning time was long due to high dimensionality.

Keywords : Distributed Processing System, Malware Detection, Feature Extraction, Machine Learning

대용량 악성코드의 특징 추출 가속화를 위한 분산 처리 시스템 설계 및 구현

이 현 종† · 어 성 율†† · 황 두 성†††

요 약

기존 악성코드 탐지는 다형성 또는 난독화 기법이 적용된 변종 악성코드 탐지에 취약하다. 기계학습 알고리즘은 악성코드에 내재된 패턴을 학습시켜 유사 행위 탐지가 가능해 기존 탐지 방법을 대체할 수 있다. 시간에 따라 변화하는 악성코드 패턴을 학습시키기 위해 지속적으로 데이터를 수집해야한다. 그러나 대용량 악성코드 파일의 저장 및 처리 과정은 높은 공간과 시간 복잡도가 수반된다. 이 논문에서는 공간 복잡도를 완화하고 처리 시간을 가속화하기 위해 HDFS 기반 분산 처리 시스템을 설계한다. 분산 처리 시스템을 이용해 2-gram 특징과 필터링 기준에 따른 API 특징 2개, APICFG 특징을 추출하고 앙상블 학습 모델의 일반화 성능을 비교했다. 실험 결과로 특징 추출의 시간 복잡도는 컴퓨터 한 대의 처리 시간과 비교했을 때 약 3.75배 속도가 개선되었으며, 공간 복잡도는 약 5배의 효율성을 보였다. 특징 별 분류 성능을 비교했을 때 2-gram 특징이 가장 우수했으나 훈련 데이터 차원이 높아 학습 시간이 오래 소요되었다.

키워드 : 분산 처리 시스템, 악성코드 탐지, 특징 추출, 기계 학습

1. 서 론

악성코드(malware, malicious software)는 컴퓨터, 서버 또는 네트워크에 손상을 주기 위해 고안된 소프트웨어로 대상 컴퓨터에 침입한 후에 사용자의 이익에 반하는 행위를 수행한다. 안티 바이러스 소프트웨어가 악성코드 공격에 대비하고 있지만 다형성(polymorphism), 변형(metamorphism), 난독화(obfuscation) 등의 변형 기술에 따라서 변종 악성코드가 존재해 실질적 대비가 어려운 실정이다[1]. 시만텍의 “2018년 인터

※ 이 논문은 2018년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임[No. 2017-0-01055, 지능화된 APT 공격의 효과적인 대응을 위한 차세대 엔드포인트 사전탐지 및 대응(EDR: Endpoint Detection and Response) 기술].

† 준 회 원 : 단국대학교 소프트웨어학과 석사과정
†† 정 회 원 : 단국대학교 소프트웨어학과 박사과정
††† 종신회원 : 단국대학교 소프트웨어학과 교수

Manuscript Received : September 07, 2018
First Revision : December 17, 2018
Accepted : December 27, 2018

* Corresponding Author : Doosung Hwang(dshwang@dankook.ac.kr)

넷 보안 위협 보고서"에 따르면 비트코인 채굴이 악성코드의 주 공격 목적이었으며 변종 랜섬웨어는 2017년 대비 40.0% 증가했고 전체 악성코드 변종은 87.7% 증가했다[2]. 변종 악성코드 공격을 방지하기 위한 대비책이 필요하다.

기존 악성코드 탐지 방법으로는 대표적으로 서명 기반 탐지 방법(signature-based detection)과 행위 기반 탐지 방법(behavior-based detection)이 있다[3-6]. 서명 기반 탐지 방법은 악성코드의 행위 규칙, 해쉬 값을 추출하고 비교분석을 통해 탐지하는 방법이다. 빠른 분석 및 탐지가 가능해 널리 사용되고 있으나, 새로운 악성코드 또는 변종 악성코드는 탐지가 어렵다. 행위 기반 탐지 방법은 악성코드에서 사용하는 함수, 라이브러리 또는 어셈블리코드 등으로부터 행위 특징을 추출하여 분석하는 방법이다. 하지만 악성 행위의 종류와 행위에 따른 위험 수준을 사전에 정의해야하는 단점이 존재한다. 기존 탐지 방법의 한계를 극복하기 위해 기계 학습을 응용한 악성코드 탐지 방법이 연구되고 있다[7-9].

악성코드 탐지를 위한 기계 학습 응용은 준비된 데이터로부터 추출된 특징 벡터에 대해 분류 규칙을 도출하거나 유사도에 근거해 분류에 적합한 파라미터를 자동 학습한다[4, 5, 10]. 학습 과정에서 숨겨진 패턴 규칙(pattern rule)을 찾아 분류 학습이 수행되어 다형성 또는 변형 기법이 적용된 변종 악성코드 탐지를 기대할 수 있다.

데이터 수집과 처리 과정에서 대용량 악성코드 데이터로부터 훈련 데이터 준비는 공간 복잡도와 시간 복잡도가 수반된다. 새롭게 나타나는 변종 악성코드를 효율적으로 저장할 수 있는 데이터베이스와 대용량 데이터를 학습 모델에 반영하는 시간을 가속화할 수 있는 수단이 필요하다. 두 조건을 만족시키기 위해서 HDFS(Hadoop Distributed File System)[11, 12] 기반 분산 처리 시스템을 설계한다.

본 논문에서는 기계 학습을 이용한 악성코드 탐지 시스템의 설계를 다룬다. 수집된 대용량 악성코드로부터 분산 처리를 이용한 특징 추출과 앙상블 모델 학습 및 평가를 수행하여 악성코드 탐지 성능을 분석한다. 2장에서는 악성코드 분석 방법과 관련 연구를 서술한다. 3장에서는 제안하는 분산 처리 시스템 설계와 처리 과정을 기술하고 학습에 사용되는 특징을 제안한다. 4장에서는 분산 처리 시스템 적용에 따른 데이터 공간 복잡도, 특징 추출 시간 복잡도를 비교하고 특징 벡터 별 일반화 성능을 분석한다. 5장에서는 결론과 향후 연구 방향을 서술한다.

2. 관련 연구

악성코드 데이터 학습에 사용되는 특징을 추출하기 위해서 정적 분석 또는 동적 분석 방법이 수행된다[3-5]. 정적 분석은 악성코드를 실행하지 않고 분석하는 방법이다. EXE(Executable) 파일의 PE(Portable Executable) 구조에서 IAT(Import Address Table)를 분석해 사용된 DLL(Dynamic Link Library)과 API (Application Programming Interface) 정보를 추출하거나 디스어셈블리를 수행해 코드 내 포함하는 문자열과 OP(Operation) 코드, API 호출 정보를 추출할 수 있다[3, 4, 13]. 동적 분석은 악성코드를 실제로 실행하고 컴퓨터의 변화를 관찰하는 방법이다. 악성코드를 직접 실행하기 때문에 독립된 가상환경을 구축해 운영 시스템에 악영향을 주는 것을 방지해야한다. 가상환경 실행 시스템으로는 CWSandbox[14], Cuckoo Sandbox[15] 등이 있으며 실행된 프로세스로부터 네트워크, 레지스트리, MFT (Master File Table) 파일, 프로세스 등의 변화를 감시하면서 로그를 기록한다.

Mansour Ahmadi 외 4명은 malware challenge 데이터[16]를 악성코드 탐지 모델 학습에 사용했다. 학습에 사용된 특징은 바이트 기반 특징과 디스어셈블리 기반 특징으로 나눠진다. 바이트 기반 특징으로는 n-gram, 엔트로피 정보, 바이트 이미지를 포함해 5개의 특징을 사용했다. 디스어셈블리 기반 특징으로는 OP 코드, 레지스트리, 특정 키워드 빈도 특징을 포함해 8개의 특징을 사용했다. 총 13개의 특징 벡터에 대한 XGBoost 모델[17]의 성능을 5차 교차 검증으로 평가했다. 엔트로피 특징이 0.98의 성능을 보였으며 특정 키워드 빈도 특징이 0.99으로 분석되었다[10].

Igor Santos 외 4명은 VX Heavens[18]로부터 17,000개의 악성코드 데이터에서 1,000개 데이터를 선택하고 정상 코드는 상용 소프트웨어 1,000개를 수집해 악성코드 데이터와 학습 데이터 집합을 구성했다. OP 코드에 대한 WTF(Weighted Term Frequency)와 2-gram 구조를 특징 벡터로 구성했다. 실험 결과로 polynomial kernel을 사용한 SVM이 0.95 분류 정확도로 성능을 보였다[13].

Steven Strandlund Hansen 외 3명은 VXHeaven[18], VirusShare[19]에서 약 270,000개의 악성코드 데이터를 제공받고 Ninite [20], Lupo PenSuite[21]에서 837개의 정상코드 데이터를 수집해 총 270,837개의 데이터로 학습 데이터 집합을 구성했다. Cuckoo sandbox[15]를 이용한 동적 분석을 통해 API 호출 정보와 호출 빈도, DLL 호출 빈도를 추출했다. 여러 개의 특징을 같이 사용함에 따라 높은 차원을 축소하기 위해 IGR (Information Gain Ratio)를 사용해 특징을 선택했다. 모델 학습 알고리즘으로 Random Forest[22]가 사용되었으며 0.98의 분류 정확도를 보였다[14].

Victoria J. Hodge 외 2명은 고차원 학습 데이터를 위한 특징 선택 알고리즘을 분산 설계했다. 분산 처리는 Hadoop Yarn 기반 프레임워크 AURA(Advanced Uncertain Reasoning Architecture)를 이용했다. 상호 정보(mutual information)기반, 상관관계 기반(correlation-based), 득실 비율(gain ratio) 기반 특징 선택과 Chi-square 알고리즘을 병렬화했다[23].

Mahfoud Bala 외 2명은 대용량 데이터를 빠르게 처리하기 위해 분산 처리 시스템 P-ETL(Prallel-ETL)을 설계했다. P-ETL 시스템의 병렬 처리는 맵리듀스(MapReduce) 방식으로 진행된다. 용량 별로 데이터를 임의로 생성해 실험에 사용했다. 300GB 데이터에 대해 맵퍼(mapper)의 수에 따른 처리 시간을 비교했다. 맵퍼의 수가 24일 때 143분, 30일 때 93분, 38일 때 87분이 소요됐다. 맵퍼의 수가 증가할수록 시간 개선 효과는 0에 수렴했다[24].

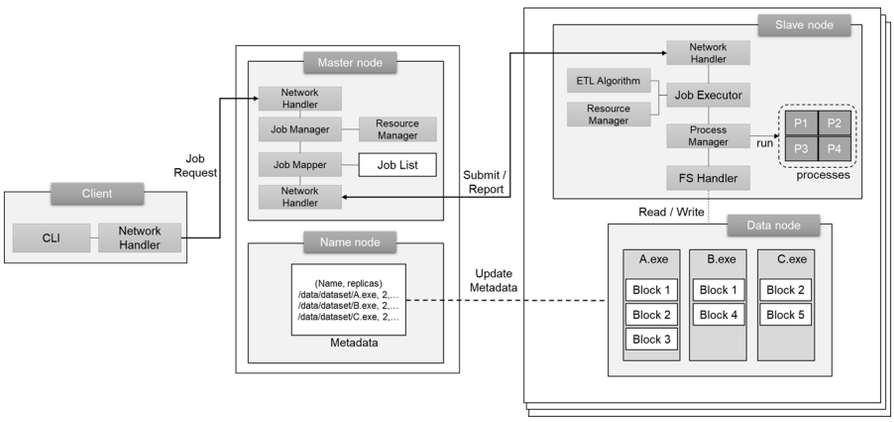

Fig. 1. Architecture of OpenDPU

3. 제안 방법

악성코드 탐지 모델을 학습하기 전에 저장된 악성코드의 특징을 추출하는 과정을 거친다. 데이터가 많을수록 증가하는 처리 시간을 가속화시키는 방법이 필요하다. ETL(Extract, Transform, Load) 작업인 특징 벡터 준비는 독립적으로 계산이 가능하여 병렬 처리가 가능하다. 다수의 스레드(thread)를 이용해 연산을 수행하면 I/O 작업에 따라 CPU가 휴식(idle) 상태로 빈번히 전환되기 때문에 최대 성능을 내지 못한다. 그러므로 분산 처리를 기반으로 IOPS(Input/Output Operations Per Second) 시간을 단축시키는 방법을 제안한다.

분산 저장된 대량의 악성코드 데이터에 대한 특징 추출 시간을 단축시키기 위해 HDFS 기반 분산 처리 시스템 Open DPU(Open Distributed Processing Unit)를 설계한다.

3.1 OpenDPU

Fig. 1은 HDFS 기반 OpenDPU의 구조이다. OpenDPU는 5대의 Intel 컴퓨터로 구성되는 클러스터로 대량의 악성코드로부터 API 함수, n-gram 등 특징벡터를 빠르게 분산 처리를 지원하는 소프트웨어 프레임워크로 설계되었다. 구성 클러스터는 마스터/슬레이브 구조이다. 마스터 노드는 작업 처리와 슬레이브 노드를 관리한다. 슬레이브 노드는 마스터 노드로부터 할당받은 작업을 요청된 프로세스에서 실행한다. 노드 간 처리 및 연결 관리 등을 위한 메시지 교환은 JSON 형식을 따른다.

OpenDPU를 이용해 데이터로부터 특징 추출하는 과정은 노드 연결, 처리 요청, 작업 분배, 작업 처리, 결과 병합 순으로 진행된다(Fig. 2).

단계 1: 노드 연결 과정에서 작업을 처리할 슬레이브 노드를 마스터 노드에 등록한다. 등록된 슬레이브 노드는 마스터 노드에게 사용가능한 리소스 정보를 공유한다.

단계 2: 처리 요청 과정에선 추출할 특징 이름과 HDFS 내 디렉토리 경로를 마스터 노드에게 전송한다. 마스터 노드는 데이터 하나를 작업(job) 하나로 간주한다. 입력된 디렉토리 내부를 순회해 DAG(Direct Acyclic Graph) 형태의 작업 목록을 생성한다.

단계 3: 작업 분배 과정에선 마스터 노드가 슬레이브 노드에게 작업을 분배한다. 작업 분배 우선순위는 할당된 작업의 수가 적은 순이다.

단계 4: 작업 처리 과정에선 슬레이브 노드가 할당 받은 작업을 처리한다. 슬레이브 노드가 HDFS에서 처리할 데이터를 가져온 후 특징 추출 알고리즘을 실행한다. 작업이 완료되면 HDFS에 중간 결과를 저장한 후 마스터 노드에게 작업 완료를 보고한다. 마스터 노드는 작업 완료 보고를 받을 때마다 작업 목록의 작업들이 모두 완료되었는지 검사한다.

단계 5: 결과 통합 과정에선 모든 작업이 완료 됐을 시 중간 결과를 병합한다. 마스터 노드는 하나의 슬레이브 노드를 지정해 데이터 내용을 병합한다.

병합된 결과는 HDFS에 저장된다. 마스터 노드는 클라이언트에게 병합 완료를 보고한다.

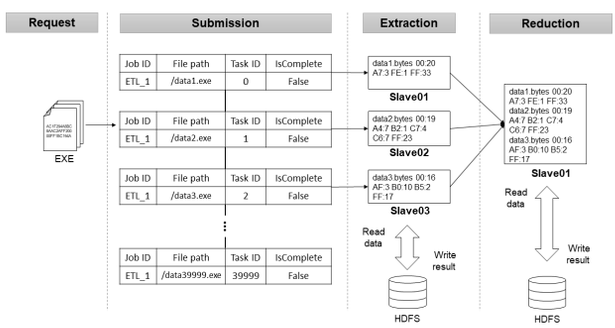

Fig. 2. Data processing steps of OpenDPU

3.2 악성코드 탐지를 위한 특징

데이터로부터 정적 분석을 이용해 16진수 바이트 정보와 어셈블리코드를 추출한다. 디스어셈블러 툴은 radare2를 사용한다[25]. 16진수 바이트에서 n-gram 특징을 추출하고 어셈블리 코드로부터 API와 APICFG(API Control Flow Graph) 특징을 추출한다.

n-gram 특징은 문장 분석, DNA 서열 분석 등 연관 관계 분석을 위해 주로 사용되는 방법으로 순서가 있는 데이터를 대상으로 분석이 수행된다[26]. 전체 데이터에서 기본 단위의 n개로 구성된 부분 데이터를 순서대로 발생시켜 빈도수를 계산해 특징 벡터를 생성한다.

n-gram의 단위는 사용자가 정의하며 악성코드 분석에서는 일반적으로 바이트를 기본 단위로 사용하며 *n*-gram의 파라미터 *n*은 2로 설정한다. 부분 데이터는 (00,00)부터 (FF,FF) 범위 내의 조합으로 발생한다[27].

API 호출 특징은 프로그램이 실행하면서 호출하는 API의 순서 정보를 나타내며 어셈블리 코드에서 call 명령어에 의해 호출된다[3]. API 호출 빈도를 계산해 특징 벡터로 사용한다. 사용자가 정의한 API가 존재하기 때문에 특징의 차원이 커질 수 있다. 따라서 학습에 사용될 API를 선정하는 작업이 필요하다.

Fig. 3은 어셈블리 코드 내에서 API 호출문을 보여준다. DLL과 API가 명시되어 있어 필터링 기준으로 사용할 수 있다. 악성코드에서 자주 사용되는 DLL 정보 또는 API 정보로 필터링해 특징 벡터를 구성한다.

APICFG 특징은 API 호출 관계를 표현한다. API 호출 정보에서 *2*-gram 구조를 추출한다. *2*-gram 구조는 API 호출 관계를 나타내며 2차원 행렬로 변환한다(Fig. 4). 2차원 행렬의 행과 열의 크기는 API 종류 수와 같아지기 때문에 API의 종류를 제한할 필요가 있다. 악성코드에서 자주 사용되는 API 정보로 필터링한 API 특징을 사용한다.

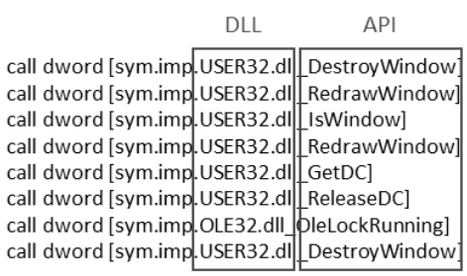

Fig. 3. API Call Statement in Assembly Code

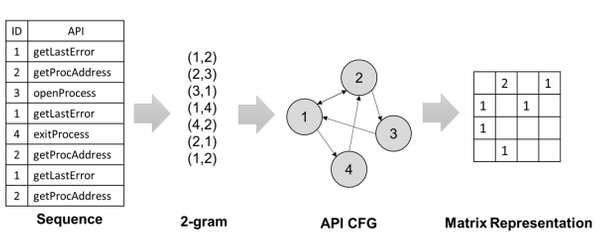

Fig. 4. Example of APICFG

4. 실험 결과

악성코드 분류 모델을 구축하기 위해 멀웨어스닷컴[27]으로부터 악성코드 20,000개와 정상코드 20,000개를 확보했다. 데이터의 클래스 레이블은 악성코드와 정상코드로 정의해 이진 분류 문제로 학습 데이터 집합을 구성했다.

분산 처리 실험에 사용된 컴퓨터는 총 5대이며 마스터 노드인 컴퓨터는 CPU Intel Core i3-3220@3.30GHz, RAM 4GB, HDD 476GB이며 슬레이브 노드는 CPU Intel Core i3-3220@2.93GHz, RAM 4GB, HDD 476GB으로 구성되었고 운영체제는 Linux Mint를 사용한다. 분산 처리 작업에서 마스터 노드는 슬레이브 노드 역할을 같이 수행한다.

Fig. 5는 *2*-gram 특징 추출 시 데이터 수에 따라 요구되는 저장 용량을 보여준다. 클러스터의 공간 복잡도는 하나의 노드에서 요구되는 저장 용량의 평균치로 계산됐다. 데이터 수가 증가할수록 필요 용량은 선형적으로 증가된다. 5대의 컴퓨터 클러스터에서 데이터를 분산시켜 저장할 경우 약 5배의 저장 효율을 보였다.

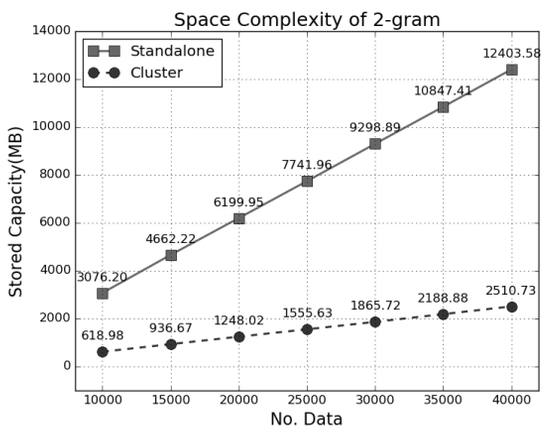

Fig. 5. Space Complexity of 2-gram Features

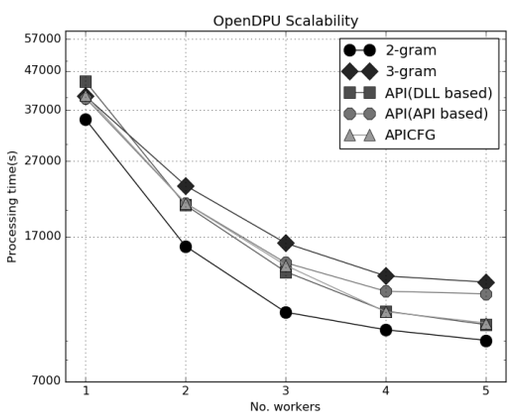

Fig. 6. Scalability of OpenDPU

Table 1은 추출된 특징의 차원 수와 노드 수에 따른 처리 시간이다. *3*-gram의 차원의 경우 최대 16,777,216 차원으로 구성되어 학습 시간 복잡도가 높아 원활한 학습을 기대할 수 없다. Fig. 6은 노드 수에 따른 처리 시간의 감소율을 보여준다. 슬레이브 노드 수 증가에 따라 감소한다. 컴퓨터 한 대의 처리 시간과 클러스터를 이용한 분산 처리 시간을 비교했을 때 평균 3.75배 가속화되었다. *2*-gram의 경우 3.87배, API의 경우 4.43배만큼 추출 속도가 개선되었다. 그래프에서 노드

Table 1. Processing Time Per Features

Features	No. dimension	Processing Time(s)				
		Single node	2 nodes	3 nodes	4 nodes	5 nodes
2-gram	61,952	34,941	16,049	10,709	9,614	9,012
3-gram	~16,777,216	40,242	23,272	16,353	13,373	12,871
API(DLL)	291	44,033	20,633	13,727	10,781	9,924
API(API)	119	39,781	20,895	14,497	12,177	11,985
APICFG	2,711	40,404	20,886	14,273	10,744	10,024

수를 늘려도 시간이 크게 단축되지 않는 현상을 보였다. 노드 수가 증가할수록 클러스터 내에서 주고받는 데이터와 메시지에 따른 네트워크 지연이 현상의 원인으로 분석된다.

특징의 일반화 성능을 비교하기 위해 XGBoost[17]와 Random Forest 학습 알고리즘[22]을 사용했다. 실험에 사용된 두 알고리즘의 파라미터는 트리 수 50, 최대 트리 깊이 15으로 설정했으며 평가로 5-식 교차 검정(5-way cross validation)을 사용했다.

Table 2는 특징 별 학습 데이터에 대한 성능과 테스트 데이터에 대한 성능이다. API 특징은 필터링 기준에 따라서 2개로 나눠 실험을 진행했다. 분류 실험 결과로 *2*-gram, API(DLL), API(API), APICFG 순으로 성능이 좋았으며 *2*-gram으로 학습한 모델의 성능이 0.94 이상으로 가장 높았지만 차원이 커 학습 시간이 가장 오래 소요됐다.

Table 2. Performance Comparison

Feature		XGBoost	Random Forest
2-gram	Train	1.00	0.98
	Test	1.00	0.94
	Time	2370.19	137.56
API (DLL)	Train	0.84	0.95
	Test	0.83	0.92
	Time	90.02	1.66
API (API)	Train	0.80	0.91
	Test	0.78	0.89
	Time	4.15	0.74
APICFG	Train	0.79	0.84
	Test	0.77	0.83
	Time	72.40	5.00

모든 특징에 대해서 학습 데이터와 테스트 데이터에 대한 성능 차이가 0.4 이하로 나타났다. 또한 두 모델을 비교했을 때 차원이 큰 특징에서는 XGBoost 모델이 성능이 우수했지만 차원이 작을 경우 Random Forest 모델이 성능이 우수하며 학습 시간이 더 적게 소요됐다.

5. 결 론

본 논문에서는 OpenDPU를 설계해 데이터 훈련 특징 처리를 가속화시켰다. 특징 추출 속도는 평균 3.75배 단축시켰다. 추출된 특징에 대해 특징 벡터를 구성하고 XGBoost와 Random Forest 모델을 학습해 분류 정확도와 학습 시간을 비교했다. *2*-gram 특징은 API 특징보다 분류 성능이 우수했지만 최대 65,536 차원으로 구성되기 때문에 시간 복잡도가 더 높게 나타났다. 고차원 특징을 저차원으로 축소할 수 있는 방법이 필요하다.

분산 처리 실험에서 노드를 늘릴수록 처리 시간이 단축되었으나 시간 단축 비율도 감소했다. Fig. 5의 그래프에 의하면 노드 수가 6대 이상으로 구성될 경우 가속화 배율이 매우 낮아져 더 이상 시간 단축 효과를 기대할 수 없다. 이에 따라 기계 학습 모델 준비 시간을 단축시키기 위해 특징 추출 분산 처리를 포함하는 모델 학습의 병렬화가 필요하다.

References

[1] I. You and Y. Kangbin. “Malware obfuscation techniques: A brief survey,” *2010 International Conference on IEEE, Broadband, Wireless Computing, Communication and Applications(BWCCA)*, 2010.

[2] Symantec, “Internet Security Threat Report,” vol.23, 2018.

[3] Michael Sikorski and Andrew Honig, “Practical Malware Analysis,” San Francisco: No Strach Press, 2012.

[4] Charles LeDoux and Arun Lakhotia, “Malware and Machine Learning,” *Intelligent Methods for Cyber Warfare, Intelligent Methods for Cyber Warfare, Studies in Computational Intelligence Book Series*, Springer, Vol.563, pp.1-42, 2014.

[5] Kaspersky Enterprise Cybersecurity, Machine Learning for Malware Detection [Internet], www.kaspersky.com/

[6] Rafiqul Islam, Ronghua Tian, Lynn M. Batten, and Steve Versteeg, “Classification of malware based on integrated static and dynamic features,” *Journal of Network and Computer Applications*, Vol.36, Issue 2, pp.646-656, 2013.

[7] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, “Novel feature extraction, selection and fusion for effective malware family classification,” in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy. ACM*, pp.183-194, 2016.

[8] I. Santos and F. Brezo, “Opcode sequences as representation of executables for data-mining-based unknown malware detection,” *Information Sciences*, Vol.231, pp.64-82, 2013.

[9] SS. Hansen, TMT. Larsen, and M. Stevanovic, “An approach for

detection and family classification of malware based on behavioral analysis," *Computing, Networking and Communications(ICNC), 2016 International Conference on. IEEE*, pp.1-5, 2016.
[10] M. Wagner, F. Fischer, R. Luh, A. Haberson, A. Rind, D. A. Keim, and W. Aigner, "A Survey of Visualization Systems for Malware Analysis," in *EG Conference on visualization (EuroVis)-STARs*, pp.105-125, 2015.
[11] Hadoop MapReduce [Internet], http://hadoop.apache.org/
[12] T. White, "Hadoop: The Definitive Guide: Storage and Analysis at the Internet Scale," 4th ed., Beijing: O'Reilly Media, 2015.
[13] C. Lin, N. Wang, H. Xiao, and C. Eckert, "Feature Selection and Extraction for Malware Classification," *Journal of Informations Science and Engineering*, Vol.31, No.3, pp.965-992, 2015.
[14] CWSandbox [Internet], https://cwsandbox.org/
[15] Cuckoo Sandbox [Internet], https://cuckoosandbox.org/
[16] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft Malware Classification Challenge," *arXiv:1802.10135v1*, 2018.
[17] T. Chen and C. Guestrin. "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM*, pp.785-794, 2016.
[18] VXHeaven [Internet], http://83.133.184.251/virensimulation.org/
[19] VirusShare [Internet], https://virusshare.com/
[20] Ninite, Ninite [Internet], https://ninite.com/
[21] Lupo PenSuite Collections, Lupo pensuite collections [Internet], http://www.lupopensuite.com/collection.htm, 2015.
[22] A. Liaw and M. Wiener, "Classification and regression by randomForest," *R news*, Vol.2, pp.18-22, 2002.
[23] V. Simon, S. O'Keefe, and J. Austin, "Hadoop neural network for parallel and distributed feature selection," *Neural Networks 78*, pp.24-35. 2015.
[24] M. Bala, O. Boussaid, and Z. Alimazighi, "P-ETL: Parallel-ETL based on the MapReduce paradigm," *Computer Systems and Applications (AICCSA), 2014 IEEE/ACS 11th International Conference on. IEEE*, 2014.
[25] Radare2 [Internet], https://rada.re/r/
[26] P. Singhal and N. Raul, "Malware detection module using machine learning algorithms to assist in centralized security in enterprise networks," *International Journal of Network Security & Its Applications(IJNSA)*, Vol.4, No.1, 2012.
[27] Malware.com [Internet], https://www.malwares.com/

이 현 종

https://orcid.org/0000-0002-2990-1545
e-mail : 72170146@dankook.ac.kr
2017년 단국대학교 컴퓨터과학과(학사)
2017년~현 재 단국대학교 소프트웨어학과 석사과정
관심분야 : Machine Learning, Parallel Processing, Image Processing

어 성 율

https://orcid.org/0000-0002-0759-6148
e-mail : secureesy@gmail.com
1997년 아주대학교 컴퓨터공학과(학사)
1999년 아주대학교 컴퓨터공학과(석사)
2018년~현 재 단국대학교 소프트웨어학과 박사과정
관심분야 : Machine Learning, Parallel Processing, Threat Intelligence

황 두 성

https://orcid.org/0000-0003-1840-9296
e-mail : dshwang@dankook.ac.kr
1986년 충남대학교 계산통계학과(학사)
1990년 충남대학교 전자계산학과(석사)
2003년 Wayne State Univ. 컴퓨터과학전공 인공지능(박사)
2003년~현 재 단국대학교 소프트웨어학과 교수
관심분야 : Machine Learning, Parallel Processing, Image Processing