PROJECT REPORT

SERVER I/O PATTERN PREDICTION BASED ON MACHINE LEARNING

머신러닝 기반 서버 I/O 패턴 예측

STUDENT NAME:	Jinseo Choi
STUDENT NUMBER:	202340213
COURSE NAME:	Data Analysis based on Machine Learning
DEPARTMENT:	Dept. of Computer engineering, College of IT Convergence
SUPERVISOR:	Prof. Ok-Ran Jeong
DATE OF SUBMISSION:	2023. 06. 1

CONTENTS

ABSTRACT

Background

AWS(Amazon web service)의 EC2 와 같은 클라우드 서비스는 운영할 때 하나의 서버를 여러 클라이언트가 사용하게 된다. 클라이언트는 지불한 금액에 따라 서버의 리소스와 접근권한을 할당 받는다. 특히 QoS(Quality of service)의 경우 특정 사용자의 성능(e.g. latency, bandwidth)를 보장받아야 한다. 그러나, 기존 QoS 동작 메커니즘은 Latency critical 클라이언트의 Latency 가 지정된 임계 값을 초과하면 Best-effort 클라이언트의 성능을 제한하여 Latency critical 클라이언트의 성능을 유지한다. 이러한 기존 방식은 Latency critical 클라이언트 성능이 임계 값을 벗어난 이후에 동작한다는 문제점이 존재한다. 이러한 문제점을 해결하기 위해 본 연구에서는 머신러닝 기법을 사용하여 서버 상 클라이언트의 I/O 패턴을 미리 예측하여 효율적으로 Latency critical 클라이언트의 성능을 보장하는 QoS 메커니즘을 제안한다.

Methods

본 연구에서는 클라이언트의 I/O 패턴을 예측하기 위해 머신러닝 모델을 사용한다. 머신러닝 모델 훈련을 위한 데이터셋으로 SNIA 에서 제공하는 MSR Cambridge I/O trace 데이터셋을 사용한다. 이를 통해 I/O 패턴을 학습하여 I/O 요청 크기를 예측하는 모델을 생성한다. 이후, 각 모델의 정확도를 측정하여 성능을 평가한다.

Evaluation

제안기법을 평가를 위해 첫째, 서로 다른 종류의 애플리케이션에 대한 I/O trace 로 학습된 3 개의 머신러닝 모델을 생성한다. 이후, 각 테스트 데이터셋을 사용하여 예측 정확도를 측정한다. 이를 통해 머신러닝 기법의 I/O 패턴 예측 가능성을 평가한다.

Discussion and Conclusion

본 연구는 머신러닝 기반 QoS 매커니즘을 구현하기 이전에 머신러닝 기법을 통한 I/O 패턴 예측 가능성에 대해 살펴본다. 향후, QoS 매커니즘을 구현하여 시뮬레이션을 통해 기존방식과 성능평가를 진행하고자 한다.

INTRODUCTION

오늘날 딥 러닝과 같은 인공지능 기법이 높은 성능을 보이면서 다양 분야에 적용되고 있다[1-4]. 딥 러닝 모델은 심층신경망(Deep neural network)으로 구성되어 있으며 대량의 데이터셋을 통해 학습을 진행한다. 이는, 높은 컴퓨팅 연산을 요구한다. 따라서 최근에는 딥 러닝 모델 훈련을 위한 클라우드 컴퓨팅 서비스가 활발하게 사용되고 있다[2]. 클라우드 컴퓨팅 서비스는 고성능 컴퓨팅 서버의 리소스를 여러 사용자가 사용할 수 있도록 리소스를 분배한다. 따라서 리소스 분배를 위한 다양한 분배정책이 존재한다.

Quality of Service

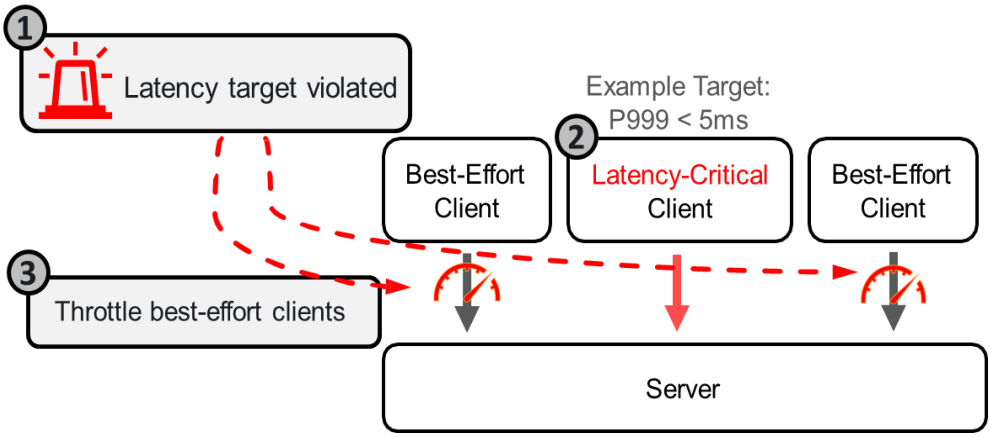


Fig.1 The example of the QoS mechanism

QoS 는 리소스 분배정책 중 하나로 특정사용자에 대해서 일정한 성능을 보장하는 분배정책이다. 크게 Critical 클라이언트와 Best-effort 클라이언트로 구분된다. Critical 클라이언트는 일정한 성능이 보장되는 클라이언트로 일반적으로 Latency 임계 값을 지정하여 임계 값 이하의 성능을 보장하도록 유지된다. Best-effort 클라이언트는 Critical 클라이언트의 성능을 보장하기 위해 성능에 제한을 받는 클라이언트이다. Fig.1 은 QoS 의 동작 예시를 보여준다. 2 개의 Best-effort 클라이언트와 1 개의 Latency critical 클라이언트로 구성된 서버가 존재할 때, Latency critical 클라이언트는 임계 값이 5ms 으로 설정되어 있다. QoS 는 클라이언트의 성능을 모니터링한다. ❶에서 Latency critical 클라이언트의 성능이 저하되는 상황이 발생할 경우 Latency critical 클라이언트의 성능이 임계 값을 넘어서게 된다❷. 이때 QoS 는 Best-effort 클라이언트의 Bandwidth 를 쓰로틀링하여 성능을 제한한다❸. 이로 인해 발생하는 유휴 Bandwidth 리소스를 Latency critical 클라이언트에 할당하여 기존 성능을 유지시킨다.

기존의 QoS 방식은 Latency Critical 클라이언트의 성능이 임계 값을 벗어난 이후에 동작한다는 문제점이 존재한다. 또한 높아진 Latency 가 다시 기준 성능을 회복하는데 시간이 소요된다. 이로 인해 일정 시간동안 성능 임계 값을 벗어난 상태로 유지된다. 이러한 문제점을 해결하기 위해 본

연구에서는 머신러닝 기법을 사용하여 클라이언트의 성능변화를 예측하여 임계 값을 벗어나기 전에 미리 쓰로틀링을 수행하여 성능을 유지하는 기법을 제안한다. 성능 변화를 예측하기 위해 우리는 I/O 패턴을 학습하여 I/O 요청 크기를 예측하는 머신러닝 모델을 설계한다.

Datasets

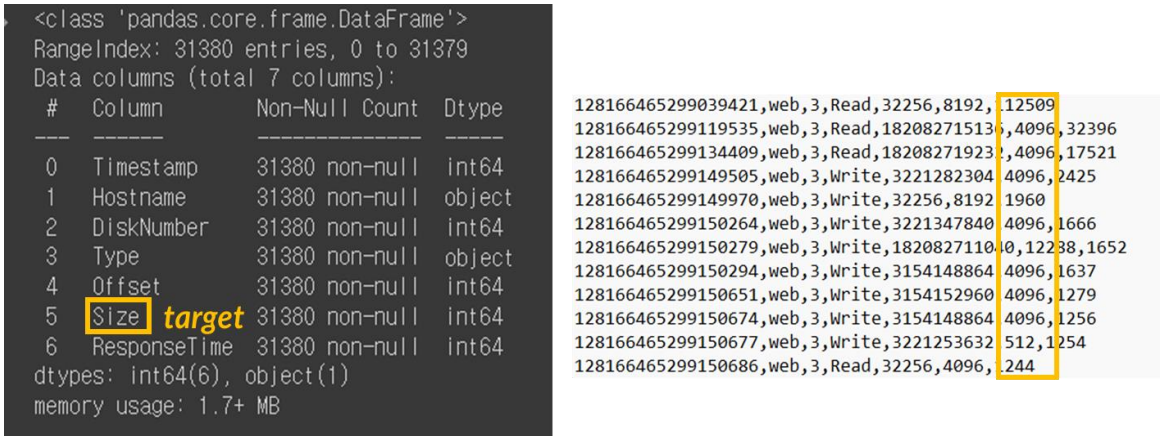


Fig.2 The I/O trace dataset description

머신러닝 모델의 훈련을 위한 데이터셋으로 우리는 SNIA 에서 제공하는 MSR Cambridge I/O trace 데이터셋을 사용한다[5]. 이는 실제 Microsoft 웹 서버에서 수집된 I/O trace 데이터셋이다. Fig.2 는 I/O trace 데이터셋의 컬럼 정보를 보여준다. 데이터셋은 I/O 를 요청한 시각인 Timestamp, I/O 를 요청한 애플리케이션을 의미하는 Host, 요청을 받은 서버 내 스토리지 번호, I/O 종류, I/O 주소 오프셋, I/O 요청 크기, 응답시간으로 구성되어 있다. 데이터셋은 여러종류의 애플리케이션에 대한 Trace 를 제공한다. 본 연구에서 제안하는 머신러닝 모델은 I/O 요청 크기를 예측하는 모델으로 설계한다.

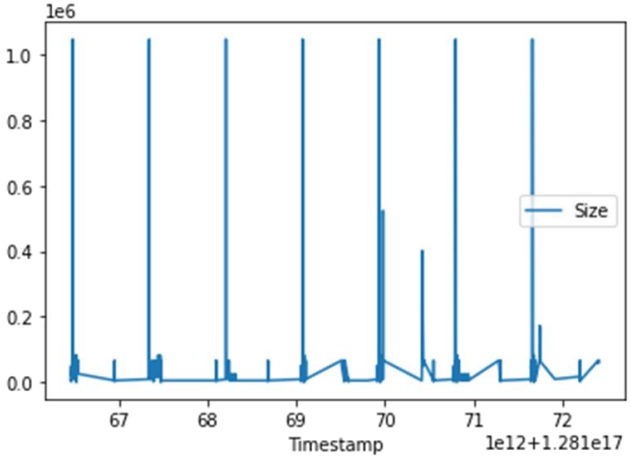


Fig.3 The example of the I/O pattern

Fig.3 은 웹 메일 애플리케이션에 대한 서버 I/O 패턴 그래프의 예시를 보여준다.

PROBLEM STATEMENT

Overview

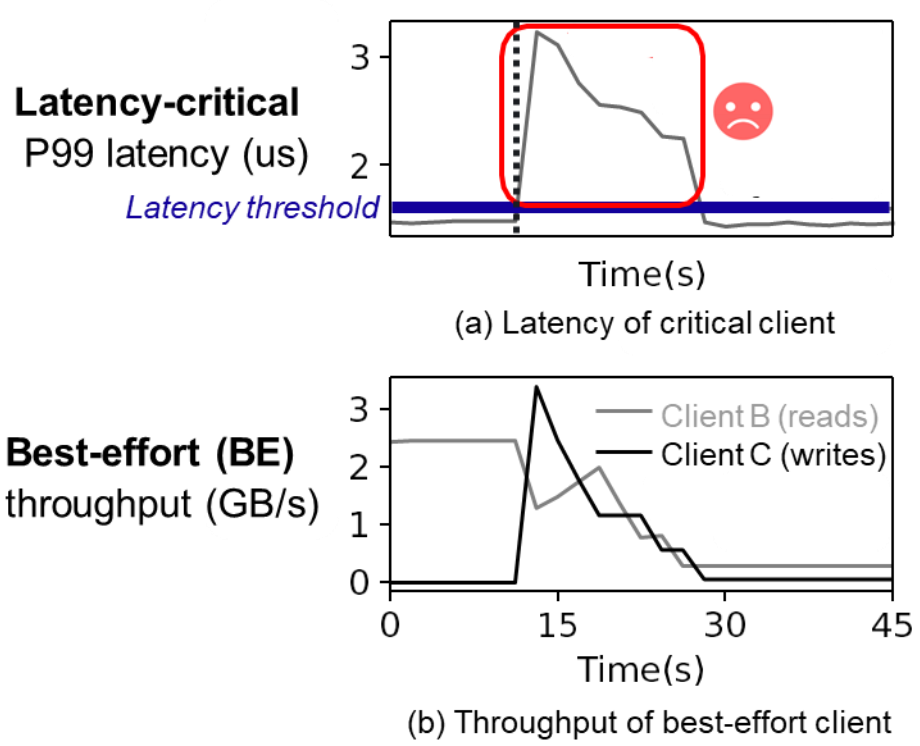


Fig.4 The example of the conventional QoS

앞서 언급한 것과 같이, 본 연구에서는 기존 QoS 의 문제점을 분석하였다. Fig. 4 는 기존 QoS 사용 시 발생하는 문제점에 대한 예시를 설명한다. (a)는 Latency critical 클라이언트의 latency 변화를, (b)는 Best-effort 클라이언트의 Throughput 변화를 보여준다. (b)에서 B 클라이언트의 Throughput 이 상승하면서 동시에 (a)의 Latency critical 클라이언트의 Latency 가 임계 값 보다 높아진다. 이후, QoS 의 메커니즘에 따라 Best-effort 클라이언트의 Bandwidth 를 제한하면서 Latency critical 클라이언트의 Latency 가 낮아진다. 이러한 동작방식은 Latency critical 클라이언트의 실제 성능이 임계 값을 넘어선 이후에 성능을 복구하기 위한 동작이 수행된다는 문제점이 존재한다. 또한 임계 값을 벗어난 이후 다시 성능이 회복될 때까지 시간이 소요된다.

Research Hypothesis

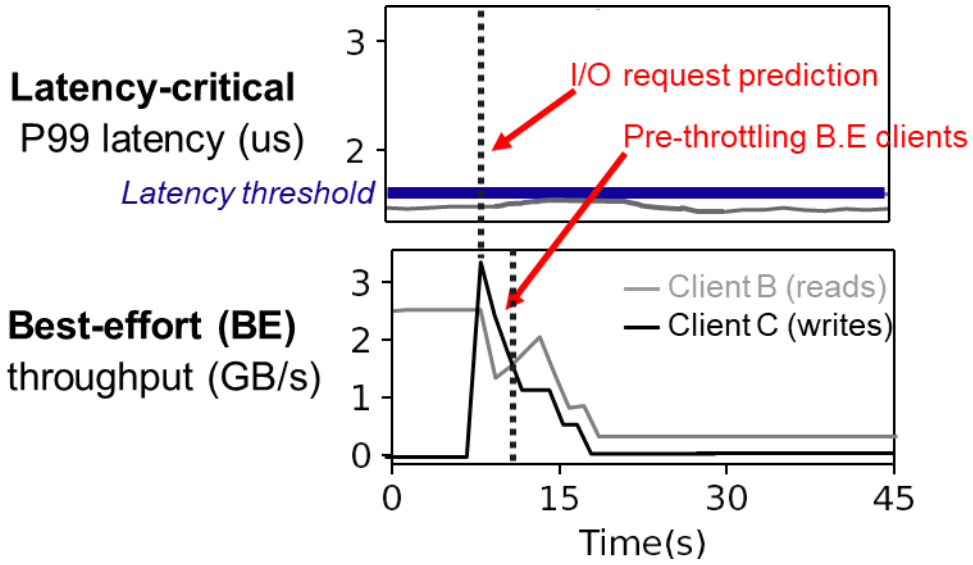


Fig.5 The hypothesis of the proposed QoS

Fig.5 는 본 연구의 제안기법인 머신러닝 기반 QoS 메커니즘을 적용했을 경우 예상되는 결과를 보여준다. 제안기법은 I/O 패턴을 학습한 머신러닝 모델을 통해 모든 클라이언트의 다음 I/O 요청 크기를 예측한다. 예측 값의 합계를 계산하여 Latency critical 클라이언트의 성능 임계 값 초과 여부를 판단한다. 임계 값을 초과할 것으로 판단할 경우, 현재 시점에서 미리 Best-effort 클라이언트의 성능을 제한한다. 이를 통해 다음 시점에서 Latency critical 클라이언트의 성능이 임계 값을 초과하는 것을 방지할 수 있다.

OBJECTIVES AND AIMS

Overall Objective

본 연구에서는 다양한 머신러닝 모델을 사용하여 성능비교를 통해 I/O 패턴 예측에 적합한 머신러닝 모델을 선택한다. Fig.2 은 모델 간 성능비교의 예시를 보여준다. 또한 다양한 튜닝기법을 적용하고 최적의 하이퍼 파라미터를 탐색하여 모델을 정확도를 향상시킨다. 이를 통해 QoS 적용 가능성을 보인다.

Specific Aims

1. RMSE(Root mean squared error)가 0.001 < RMSE < 0.01 구간의 성능을 보이는 머신러닝 모델 구현

DESIGN

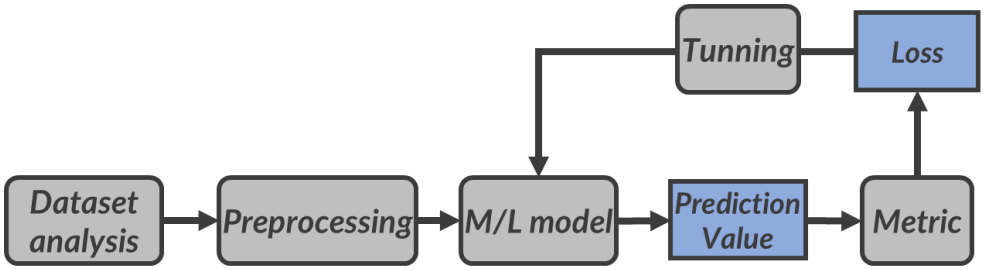


Fig.5 The development flow of machine learning model

본 연구에서는 Fig.5 와 같은 과정을 통해 예측 모델을 구성하여 실험을 진행한다. 또한 머신 러닝, 딥 러닝, 크게 2 가지 방법으로 모델을 구성하여 실험을 진행한다.

Dataset analysis

Server	Function	#volumes
usr	User home directories	3
proj	Project directories	5
prn	Print server	2
hm	Hardware monitoring	2
rsrch	Research projects	3
prxy	Firewall/web proxy	2
src1	Source control	3
src2	Source control	3
stg	Web staging	2
ts	Terminal server	1
web	Web/SQL server	4
mds	Media server	2
wdev	Test web server	4

Table 1. Data center server traced (13 servers, 36 volumes, 179 disks)

학습모델 훈련에 사용할 데이터는 Storage Networking Industry Association(이하 SNIA)에서 제공하는 MSR Cambridge trace 데이터셋을 사용한다[6]. 이는 Microsoft enterprise 의 13 개 서버에서 1 주일간 수집된 I/O 트레이스 데이터로, 36 가지 종류의 트레이스를 제공한다. 데이터는 .csv 형식으로 제공되며 I/O 요청 시간인 Timestamp, 요청한 주체인 Hostname, 스토리지 번호인 DiskNumber, 연산 종류인 Type, I/O 요청의 바이트 오프셋인 Offset, I/O 요청 크기인 Size, 응답시간인 ResponseTime, 7 가지 속성을 가진다. 우리는 Media server 트레이스 데이터를 사용하였다.

Media server trace

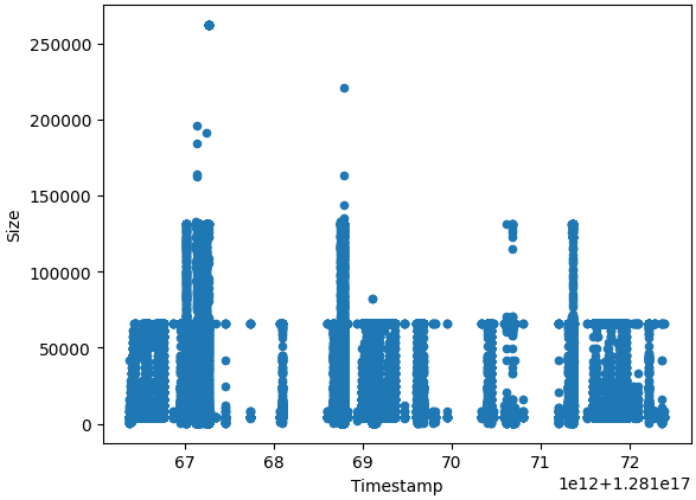


Fig.6 The relation of time and I/O request size of media server trace dataset.

Fig. 6 은 미디어 서버 트레이스의 데이터셋의 시간 흐름에 따른 I/O 요청 크기 분포를 보여준다. 데이터셋에서 I/O 요청크기는 특정 구간에 집중되어 있는 것을 확인할 수 있으며 요청 횟수가 시간 흐름에 따라 일정한 주기로 반복되는 패턴을 보인다. 이를 통해 본 연구에서 예측하고자 하는 I/O 요청 크기는 시간 흐름에 따른 시계열 예측이 가능할 것으로 생각된다. Fig. 7 은 I/O 요청 크기와 응답시간의 관계성을 나타낸 그래프이다. 응답시간은 대부분 100ns 이하, 횟수는 1.2 MB 이하에 집중되어 있다.

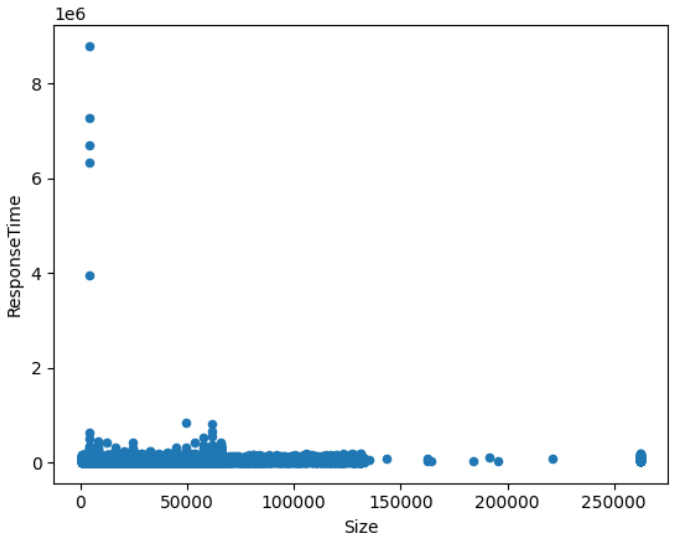


Fig.7 The relation of I/O request size and response time of media server trace dataset.

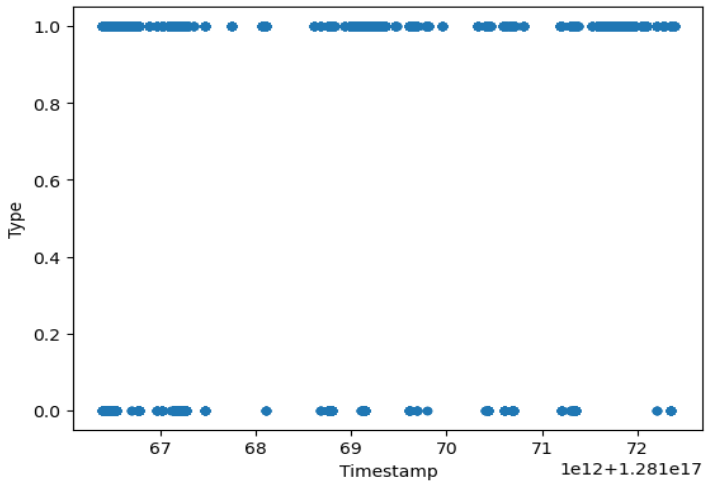


Fig.8 The relation of timestamp and type of media server trace dataset.

Fig.8 은 I/O 시간흐름에 따른 I/O 연산 종류의 분포를 보여준다 I/O 연산은 Read, Write 두가지로 표현되며 0 또는 1 으로 표현된다. 데이터셋에서는 쓰기연산 횟수가 더 많은 것을 확인할 수 있다.

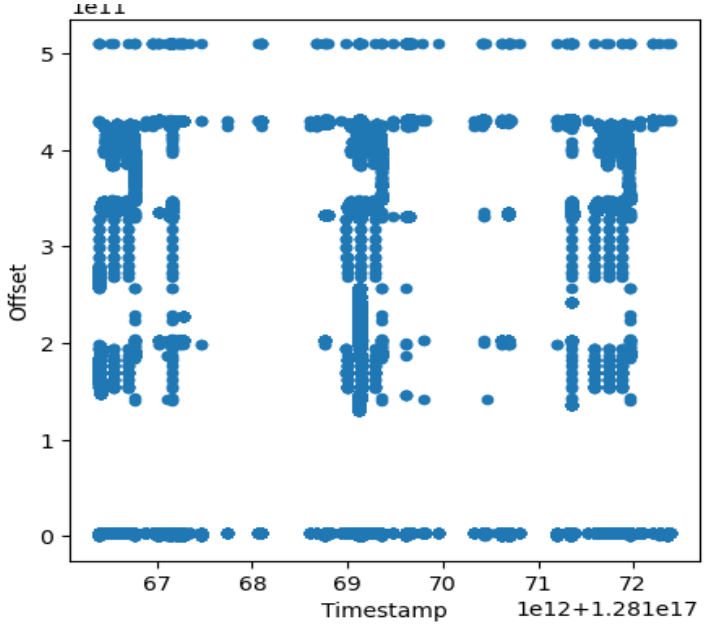


Fig.8 The relation of timestamp and offset of media server trace dataset.

Fig.9 은 I/O 시간흐름에 따른 바이트 오프셋의 크기를 보여준다. 이는 Fig.6 의 그래프와 패턴의 유사성을 보이는 것을 확인할 수 있다. I/O 요청 데이터 크기가 클수록 바이트 오프셋의 범위도 커진다.

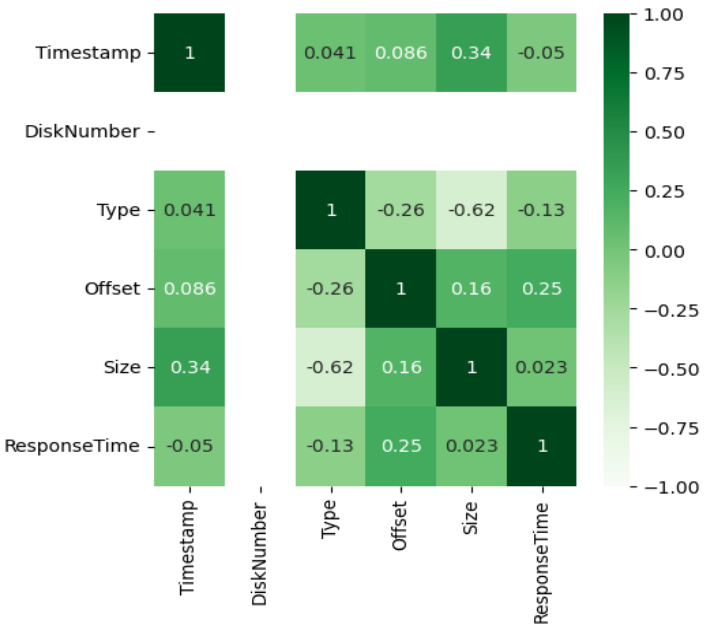


Fig.9 The heat map of each feature by calculating relation coefficient.

Fig. 9 는 데이터셋의 각 요소들의 관계성을 확인하기 위한 히트맵으로, 관계계수를 사용하여 데이터 간 관계성을 계산하여 표현하였다. Disknumber 의 경우 모든 데이터에서 동일한 값으로 표현되므로 제외하였다. Fig. 9 에서 예측목표인 Size 값과 가장 연관성이 높은 요소는 Timestamp 로, Fig.6 에서 확인한 시간 흐름에 따른 일정한 패턴으로 가장 관계성이 높은 것으로 나타났다. 다음으로 관계성이 높은 요소는 바이트 오프셋으로, Fig. 8, Fig. 6 에서와 같이 비슷한 패턴을 보임으로써 요소 간 관련성이 존재할 것으로 생각된다. 반면, Type 의 경우 낮은 값을 보인다. 이는 데이터를 읽고 쓰는 연산과 각 연산에서 요청하는 데이터 크기 간 연관성이 낮은 것을 의미한다. 본 연구에서는 Timestamp, Responsetime, Offset, 3 가지 요소를 사용하여 Size 를 예측한다.

Dataset preprocessing

```
# Input scale
colors = {0:"red", 1:"blue"}
trace.dropna(axis=0)

scaler_x = MinMaxScaler()
scaler_x.fit(trace.iloc[:, :-1])

trace.iloc[:, :-1] = scaler_x.transform(trace.iloc[:, :-1])
test_trace.iloc[:, :-1] = scaler_x.transform(test_trace.iloc[:, :-1])

# Output scale
scaler_y = MinMaxScaler()
scaler_y.fit(trace.iloc[:, [-1]])

trace.iloc[:, -1] = scaler_y.transform(trace.iloc[:, [-1]])
test_trace.iloc[:, -1] = scaler_y.transform(test_trace.iloc[:, [-1]])
```

Fig.10 The code of dataset preprocessing

데이터셋은 1637711 행으로 구성되었으며, 약 87MB 의 용량을 가진다. 앞서 살펴본 데이터 특성에서 각 요소간 값의 단위 차이가 큰 것을 확인할 수 있다. 따라서 본 연구에서는 Min-max 스케일링을 적용하여 각 요소가 0~1 의 값을 가지도록 데이터셋을 변환하였다. 또한 데이터셋 중 NaN 값을 0 으로 변경하였다.

M/L model

Machine learning

머신러닝 모델의 경우 본 연구에서는 I/O 요청 크기 값 수치를 예측해야 하므로 5 가지회귀모델을 선택하여 성능을 측정하였으며 Randomforest 기법을 적용하여 성능 최적화를 시도했다. 모델 구현은 sklearn 라이브러리를 사용하였다.

Deep learning

추가적으로 딥 러닝 기법을 적용하였다. 딥 러닝 모델은 시계열 데이터 학습에 적합한 RNN(Recurrent neural network) 기반의 LSTM(Long-short term memory)모델을 사용하였다. 딥 러닝 모델의 경우 오차역전파를 위한 옵티마이저는 Adam 을 사용하였다.

EVALUATION

Machine learning

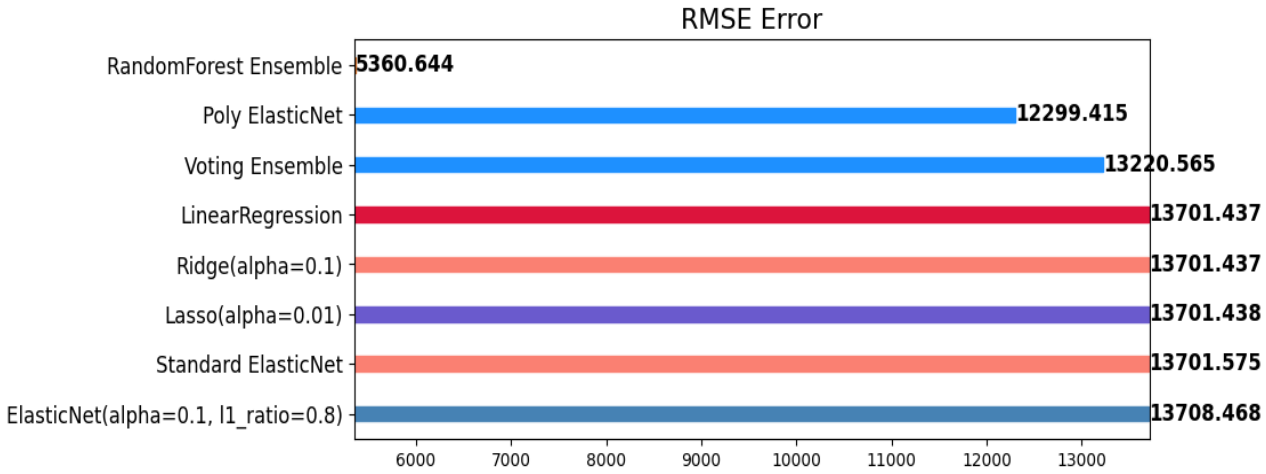


Fig.11 The result of RMSE

Fig.11 은 머신러닝 모델의 RMSE 결과값을 보여준다. Randomforest 의 경우 가장 낮은 RMSE 값을 보인다. Randomforest 의 경우 높은 1000 회 시행한 결과이다. Fig.12 는 Poly ElasticNet 과 Randomforest 의 실측값 비교결과를 보여준다. Randomforest 의 경우 예측값 분포가 실측 값에 더 집중된 것을 확인할 수 있다.

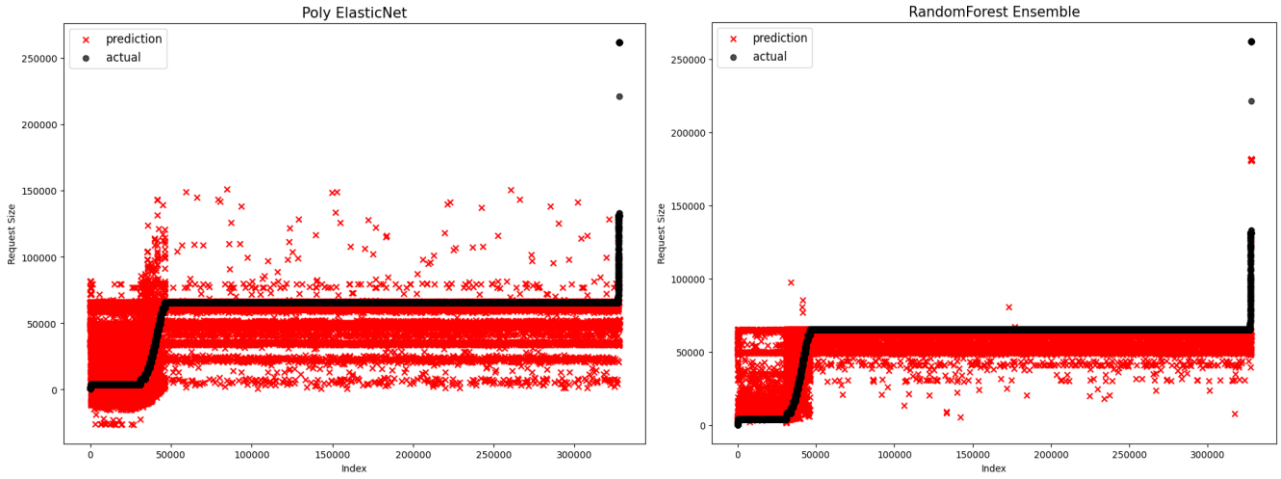


Fig.12 The result of prediction-label comparison

Deep learning

```
data_dim = 4
hidden_dim = 6
output_dim = 1
learning_rate = 0.001
nb_epochs = 100
device = torch.device("cuda")

class Net(nn.Module):
    def __init__(self, input_dim, hidden_dim, seq_len, output_dim, layers):
        super(Net, self).__init__()
        self.hidden_dim = hidden_dim
        self.seq_len = seq_len
        self.output_dim = output_dim
        self.layers = layers

        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers=layers,
                            # dropout = 0.1,
                            batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim, bias = True)

    def reset_hidden_state(self):
        self.hidden = (
                torch.zeros(self.layers, self.seq_len, self.hidden_dim),
                torch.zeros(self.layers, self.seq_len, self.hidden_dim))

    def forward(self, x):
        x, _status = self.lstm(x)
        x = self.fc(x[:, -1])
        return x
```

Fig.13 The code of LSTM model

Fig.13 은 LSTM 모델을 구현한 코드와 하이퍼파라미터 설정값을 보여준다. PyTorch 프레임워크를 사용하여 모델을 구현하였으며 LSTM 레이어와 1 개의 Fully connected 레이어를 가진다. 모델 클래스 구성 시 LSTM 히든 레이어 수를 결정할 수 있도록 구성했으며 레이어 수는 6 개로 설정했다. 입력 데이터는 4 개 컬럼데이터를 입력 받으며 I/O 요청크기인 1 개의 데이터를 출력하도록 구성했다.

Adam 의 Learning rate 는 기본값인 0.01 로 설정하였으며 오차함수는 RMSE, Epoch 은 100 으로 설정하였다. 데이터의 경우 임의로 40 개의 데이터를 학습한 후 1 개의 추론 값을 출력하도록 설정했다. 모델 훈련 시 Batch 크기는 512 로 설정하였다. Fig. 14 는 Epoch training loss 값을 보여준다. 30 epoch 이후부터 Training loss 가 0 에 거의 수렴된 상태로 진행되는 것을 확인할 수 있다. 검증 데이터에 대한 RMSE 오차는 0.0018 의 결과를 보였다. Early stop 적용 시 3 번째 Epoch 에서 종료되었다. 따라서 과적합으로 판단하고 learning rate, shuffle, dropout, 등의 하이퍼파라미터를 다양하게 조정한 후 다시 훈련을 수행하였지만 동일한 결과를 보였다.

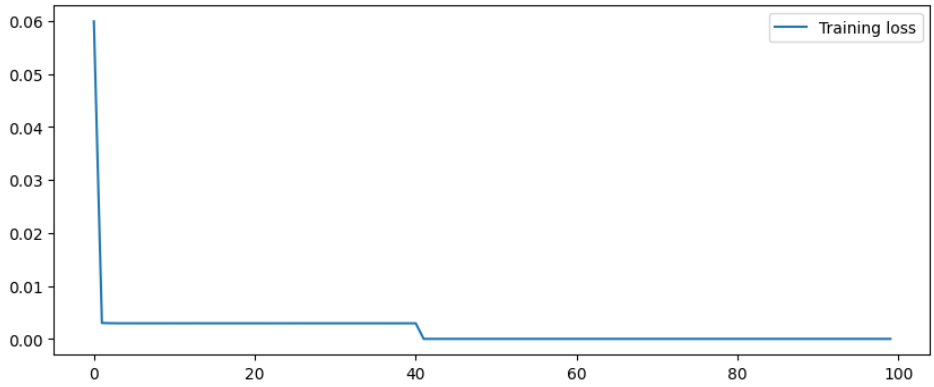
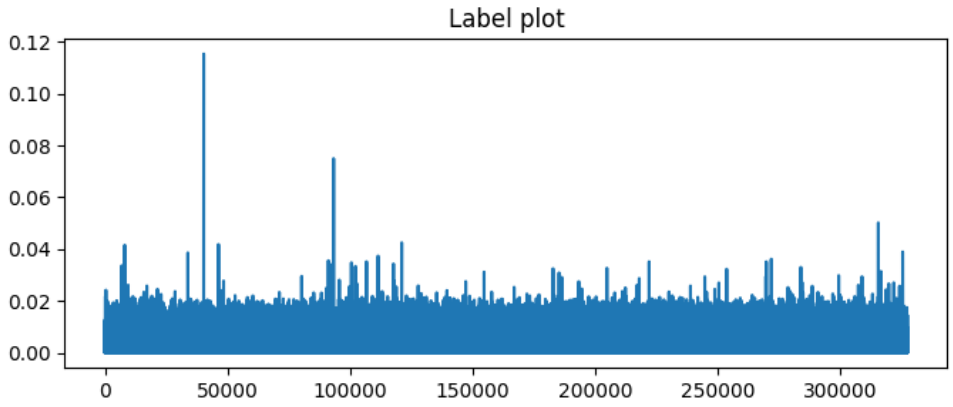
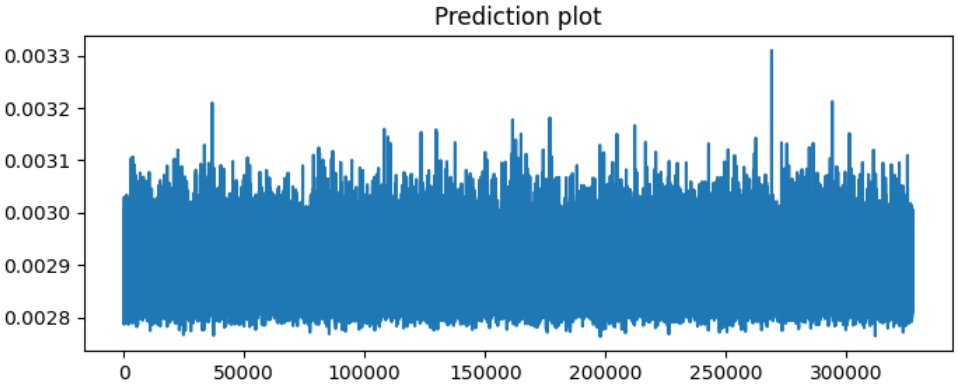


Fig.14 The result of LSTM training loss in 100 epoch



a. Test dataset label



b. The result of test dataset prediction by trained LSTM

Fig.15 The results of test data prediction

Fig.15 는 테스트 데이터셋을 사용하여 모델의 예측 값과 실측 값에 대한 결과를 보여준다. 비교결과, 모델의 예측 값 단위가 실제 값에 비해 10e-2 배만큼 낮은 값을 보이며 그 범위도 상당히 좁은 결과를 보였다. 따라서 데이터셋 분석이 미흡한 것으로 판단하여 추가적인 분석을 수행하였다.

DISCUSSION

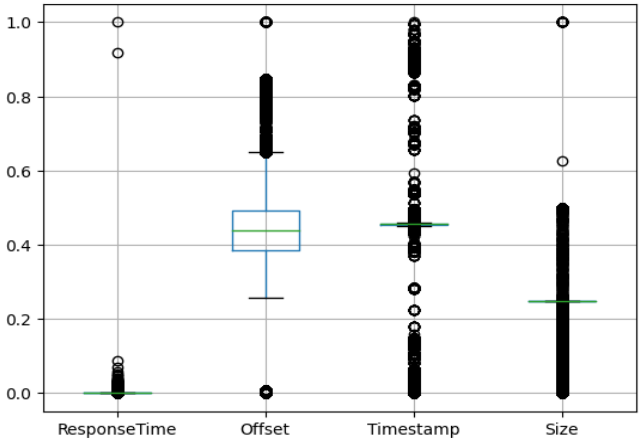


Fig.16 The box plot of min-max processed dataset

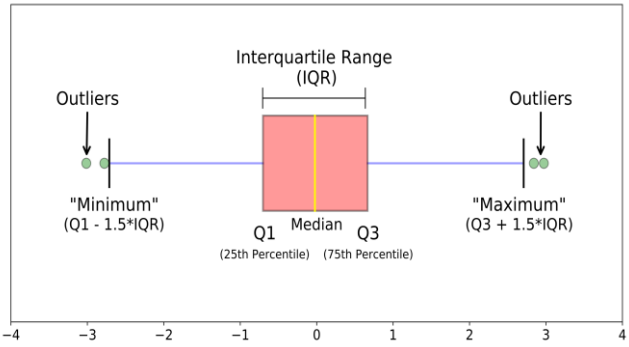


Fig.16 The principle of IQR outlier removal [7]

Fig.16 는 데이터셋 분포를 보여준다. 분석결과, 모든 feature 에서 이상치가 존재하는 것을 확인할 수 있다. 특히 ResponseTime 과 Size 의 경우 대부분의 값이 특정구간에 집중되어 있는 것을 확인할 수 있다. 따라서 이상치를 제거하기 위해 가장 간단한 방법인 IQR(Inter quantile range) 적용하였다. 적용 결과, Target feature 인 Size 값이 모두 동일한 값으로 변경된 결과를 보였다. Fig.17 은 Size 의 히스토그램을 보여준다. 80% 이상의 데이터가 특정 구간에 집중되어 있는 것을 확인할 수 있다. 이로 인해 또한 모델 훈련 시 추론 값의 구간이 좁아지는 것으로 판단된다.

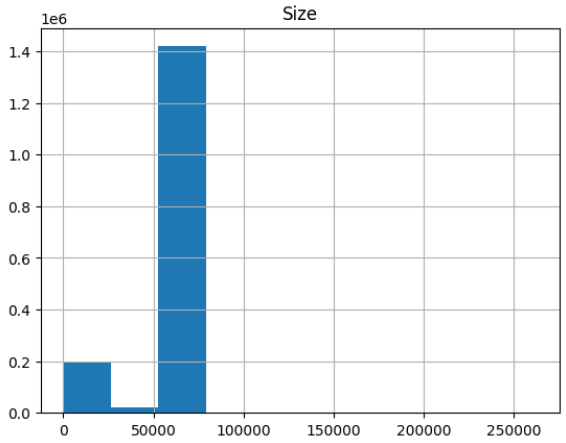


Fig.17 The histogram of target feature(‘Size’)

CONCLUSION

본 연구에서는 대용량 서버에서 QoS 와 같은 정책을 효과적으로 유지하기 위해 머신러닝 기법을 사용하여 I/O 를 예측하는 실험을 진행하였다. 이를 위해 실제 I/O 트레이스 데이터셋을 사용하였으며, 머신러닝 기법 중 회귀모델을 사용하였으며, 여러가지 최적화 기법을 적용하였다. 또한 딥 러닝 기법 중 시계열 데이터 학습에 적합하다고 알려진 LSTM 모델을 사용하여 훈련 및 학습을 진행하였다. 그 결과 검증 데이터 RMSE 가 0.0018 의 값을 보였다. 다만, 훈련 과정에서 과적합 되는 경향을 보였으며, 실측데이터와 예측 값 비교 결과 데이터 분포 경향은 비슷하나 출력 값 차이가 큰 결과를 보였다. 이에 추가적인 데이터셋 분석을 진행한 결과 Target feature 인 데이터가 지나치게 특정 구간에 편중되어 있음을 확인할 수 있었다. 향후, 비교적 균일한 분포를 보이는 데이터셋 탐색하고 IQR 이상치 제거기법 이외 다양한 전처리 과정을 통해 예측 성능을 높일 수 있을 것으로 기대된다.

REFERENCES

[1] Hao, M., Toksoz, L., Li, N., Halim, E. E., Hoffmann, H., & Gunawi, H. S. (2020, November). LinnOS: Predictability on Unpredictable Flash Storage with a Light Neural Network. In *OSDI* (pp. 173-190).

[2] YEUNG, Gingfung, et al. Towards GPU utilization prediction for cloud deep learning. In: *Proceedings of the 12th USENIX Conference on Hot Topics in Cloud Computing*. 2020. p. 6-6.

[3] OLY, James; REED, Daniel A. Markov model prediction of I/O requests for scientific applications. In: *Proceedings of the 16th international conference on Supercomputing*. 2002. p. 147-155.

[4] AGARWAL, Megha, et al. Active learning-based automatic tuning and prediction of parallel i/o performance. In: *2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW)*. IEEE, 2019. p. 20-29.

[5] Campello, Daniel, et al. Filesystem SysCall Traces SNIA IOTTA Trace Set 5198, SNIA IOTTA Trace Repository, http://iotta.snia.org/traces/system-call?only=5198

[6] Narayanan, Dushyanth, Austin Donnelly, and Antony Rowstron. "Write off-loading: Practical power management for enterprise storage." *ACM Transactions on Storage (TOS)* 4.3 (2008): 1-23.

[7] https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51