IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

2021년 8월 13일 접수, 2021년 8월 31일 승인, 2021년 9월 3일 발행, 2021년 9월 16일 현재 버전 발행.

디지털 객체 식별자 10.1109/ACCESS.2021.3110291

Kafka-ML을 통해 분산 및 심층 신경망 모델 관리 및 배포하기 클라우드-사물 간 연속성

알레한드로 카르네로, 크리스티안 마르틴, 다니엘 R. 토레스, 다니엘 가리도, 마누엘 디아즈, 바르톨로메 루비오
ITIS 소프트웨어 연구소, 말라가 대학교, 29016 말라가, 스페인

교신저자: 크리스티안 마르틴(cmf@lcc.uma.es)

이 작업은 Grant RT2018-099777-B-100에 따른 스페인 프로젝트 'rFOG: 중요 서비스를 위한 FOG로의 오프로드된 계산의 지연 시간 및 안정성 개선', "IntegraDos'의 지원을 받았습니다: 클라우드 센서 통합을 통한 사물 인터넷 실시간 서비스 제공"을 그랜트 PY20_00788에, '포그의 딥 러닝 서비스 기반 고급 모니터링 시스템"을 그랜트 UMA18FEDERJA-215에 지원했습니다.

개요 사물 인터넷(IoT)은 지속적으로 성장하여 물리적 세계 정보를 모니터링하기 위한 중단 없는 데이터 스트림 파이프라인을 생성하고 있습니다. 따라서 인공 지능(AI)은 지속적으로 진화하여 삶의 질과 비즈니스 및 학술 활동을 개선합니다. Kafka-ML은 프로덕션 시나리오에서 데이터 스트림을 통해 머신 러닝(ML) 및 AI 파이프라인을 관리하는 데 중점을 둔 오픈 소스 프레임워크입니다. 따라서 실제 애플리케이션에서 심층 신경망(DNN) 배포를 용이하게 합니다. 하지만 이 프레임워크는 클라우드-사물 간 연속체에서 DNN 모델의 배포를 고려하지 않습니다. 분산 DNN은 서로 다른 인프라 간에 계산 및 네트워크 부하를 할당하여 지연 시간을 크게 줄입니다. 이번 작업에서는 클라우드-사물 간 연속체 전반에 걸쳐 분산 DNN의 관리 및 배포를 지원하기 위해 Kafka-ML 프레임워크를 확장했습니다. 또한, 예측 시 즉각적인 대응을 제공하기 위해 클라우드-사물 간 레이어에 조기 종료 기능을 포함할 가능성도 고려했습니다. 저희는 세 가지 다른 클라우드-사물 시나리오에서 DNN 모델 AlexNet을 조정하고 배포하여 이러한 새로운 기능을 평가했습니다. 실험 결과, Kafka-ML은 클라우드 전용 배포에 비해 클라우드-사물 연속체 전체에 DNN 모델을 배포함으로써 응답 시간과 처리량을 크게 개선할 수 있음을 입증했습니다.

인덱스 용어 분산형 심층 신경망, 데이터 스트림, 클라우드 컴퓨팅, 포그/엣지 컴퓨팅, 머신 러닝, 인공 지능.

I. 소개

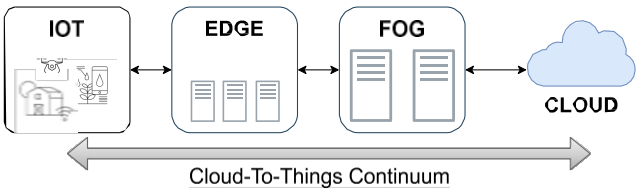
지난 몇 년 동안 수많은 출처와 분야에서 수많은 데이터를 지속적으로 확보해 왔습니다[1]. 머신러닝(ML)과 인공지능(AI)[2] 알고리즘은 이러한 데이터를 처리하는 데 필수적인 역할을 담당해 왔으며, 컴퓨터화 시대에 실용적인 도구에 의존할 수 있게 되었습니다. 유용한 예측과 제안을 통해 비즈니스 활동과 사람들의 삶이 획기적으로 개선되었기 때문에 이는 매우 바람직한 현상입니다. 예를 들어, 잘 알려진 소셜 미디어 기업들은 방대한 양의 정보를 분석하여 부적절한 콘텐츠로 간주되는 것을 감지합니다 [3], [4]. 또한 의료 [5], [6], 구조적
건강 모니터링[7], [8], 교육 시스템[9] 등 다양한 분야에서 활용되고 있습니다.

이 원고의 검토를 조정하고 출판을 승인한 부편집장은 Qing Yang 이었습니다.

등의 기술도 이러한 발전 덕분에 발전했습니다. 그럼에도 불구하고 이 모든 것은 정확한 답변을 제공하는 데 오랜 시간이 걸리는 지속적인 데이터 처리를 의미하며, 이는 대부분의 실시간 사례에서 충분하지 않을 수 있습니다.

최근에는 사물 인터넷(IoT)[10]이 데이터 스트림 생성의 대부분을 담당하고 있습니다. 실제로 이러한 데이터 소스의 수가 증가하고 있으며 2030년에는 관련 기기가 5,000억 개에 달할 것으로 예상됩니다[11]. 이는 향후 몇 년 동안 헤아릴 수 없을 정도로 많은 입력 소스와 필요한 응답 시간으로 인해 현재의 ML/AI 알고리즘이 어려움을 겪게 될 것임을 의미합니다. 이러한 알고리즘에 의존하는 대부분의 시스템은 휘발성, 증가, 연속적인 데이터 스트림이 아닌 지속적이고 정적인 정보로 작동합니다. 따라서 제어가 불가능할 뿐만 아니라 향후 수년간 큰 어려움을 겪을 수 있습니다.

데이터 소스뿐만 아니라 적응성 문제도 있습니다. 따라서 데이터 스트림 기술은 현재 이러한 도구에 대한 지원이 제한적인 Tensorflow [13] 및 PyTorch [14]와 같은 뛰어난 프레임워크[12]에서 고려해야 합니다.



이전의 문제를 극복하기 위해 데이터 스트림에서 ML/AI 파이프라인을 관리할 수 있는 오픈소스[1] 데이터 스트림을 통해 ML/AI 파이프라인을 관리하기 위한 프레임워크입니다. Kafka-ML은 데이터 스트림 디스패치를 위해 널리 사용되는 분산형 메시지 시스템인 아파치 카프카[16]를 기반으로 하며, 프로덕션 환경에 배포하기 위해 마이크로서비스 및 컨테이너화 아키텍처를 채택하고 있습니다. 추론 단계에서 Kafka-ML의 ML/AI 알고리즘은 단일 처리 장치, 즉 전체 ML 모델로 배포할 수도 있지만, 내결함성과 고가용성을 위해 클러스터에 분산하여 배포할 수도 있습니다. 하지만 짧은 지연 시간이 필요한 상황에서는 정보가 생성되는 곳(예: 엣지 또는 포그)과 최대한 가까운 곳에서 더 빠른 응답을 하는 것이 바람직합니다. 클라우드 컴퓨팅과 같은 패러다임이 제공하는 지연 시간이 저지연 애플리케이션의 요구 사항을 충족하지 못할 수 있다는 것은 의심의 여지가 없으며, 따라서 엣지 컴퓨팅과 포그 컴퓨팅[17]이 이러한 사용 사례에 기여하고 있습니다. 또한 ML/AI 알고리즘, 특히 심층 신경망(DNN)은 일반적으로 많은 컴퓨팅 리소스와 특수 하드웨어(예: GPU)가 필요하므로 리소스가 제한된 시스템의 요구 사항을 충족하지 못할 수 있습니다. 이러한 이유로 DNN은 더 작은 처리 장치(및 하위 모델)를 분산하기 위해 클라우드-사물 연속체를 따라 파티셔닝됩니다. 클라우드-사물 연속체(그림 1)는 IoT와 클라우드 사이에 위치한 포그 서버 및 엣지 디바이스와 같은 처리 장치의 집합으로 정의할 수 있습니다. 이러한 처리 장치는 시간에 민감한 애플리케이션의 대역폭 소비와 응답 시간을 최적화합니다. 예를 들어, 이러한 맥락에서 가능한 배포는 데이터 스트림을 생성하는 IoT 디바이스를 엣지 디바이스에 연결한 다음, 생성된 정보를 클라우드로 전송하기 전에 중간 처리를 위해 포그 서버에 연결된 엣지 디바이스를 배치할 수 있습니다. 이러한 처리 장치는 하위 모델이 가장 낮은 계층에서 정확도가 떨어질 수 있지만 응답 시간을 단축하는 DNN을 할당할 수 있습니다[18]. 이 패러다임은 분산형 심층 신경망(DDNN)으로 알려져 있습니다[19].

그림 1. 클라우드-사물 연속체.

프레임워크인 Kafka-ML은 DNN과 함께 작동하도록 준비된 것이지 DDNN 애플리케이션과 함께 작동하도록 준비된 것이 아니기 때문에, 이 백서에서는 수명 주기에서 DDNN을 사용할 수 있게 함으로써 Kafka-ML의 기능을 확장하는 확장 프로그램을 소개합니다.

[1] https://github.com/ertis-research/kafka-ml

이 새로운 버전은 DDNN 애플리케이션의 전체 수명 주기 파이프라인과 데이터 스트림 관리 등 Kafka-ML과 동일한 기능을 제공합니다. 또한, 클라우드-투-사물 컨티뉴엄의 분산 및 이기종 클러스터에서 지연 최적화를 위한 DDNN 모델 훈련, 통합 또는 조기 종료, 하위 모델 배포를 지원합니다. 15]에서는 Kafka-ML 프레임워크를, [18]에서는 클라우드-사물 연속체에서 DDNN을 연결하기 위한 아키텍처를 발표했습니다. 이번 연구에서는 클라우드-사물 간 컨티뉴엄의 여러 계층에서 DDNN 애플리케이션에 대한 추가 평가와 더불어 DDNN 애플리케이션을 활성화하고 수명 주기를 관리하기 위한 Kafka-ML의 확장에 중점을 둡니다.

이 글의 나머지 부분은 다음과 같이 구성되어 있습니다. 섹션 II에서는 관련 작업을 소개합니다. 섹션 III에서는 이 작업을 수행하게 된 동기를 설명합니다. 섹션 IV에서는 분산 모델의 설계를 설명합니다. 섹션 V에서는 카프카-ML의 분산 ML 모델의 ML/AI 파이프라인을 자세히 설명합니다. 그런 다음, 섹션 VI에서는 분산 ML 모델을 위한 Kafka-ML 아키텍처와 그 구성 요소에 대해 설명합니다. 섹션 VII에서는 프레임워크 평가를 위해 수행한 검증 결과를 보여줍니다. 마지막으로, 섹션 VIII에서는 결론과 향후 가능한 작업을 제시합니다.

II. 관련 작업

쿠버네티스용 오픈소스 머신 러닝 툴킷인 쿠브플로우[20]는 하이퍼파라미터 및 전처리와 같은 여러 단계의 머신 러닝 및 인공지능 파이프라인을 구성할 수 있습니다. 그럼에도 불구하고 클라우드-사물 간 연속성 및 데이터 스트림을 통한 DDNN에 대한 유연한 지원은 아직 이 솔루션 설계에서 빠져 있습니다. 에지렌즈[21]와 헬스포그[22]는 프레임워크로서 엣지-포그-클라우드 환경에서 딥 러닝 기반 애플리케이션을 배포하고 서비스 품질(QoS)을 개선합니다. 이 두 가지 프레임워크는 모두 분산형 머신 러닝 모델을 실행하며, DDNN을 연속체에 맞게 조정하지 않습니다. 또한 에지렌즈는 해상도를 축소하여 지연 시간을 줄입니다. 내결함성 아키텍처인 IoTEF[23]는 클라우드와 엣지 클러스터를 통합된 방식으로 관리하고 모니터링합니다. 그러나 내결함성을 넘어선 짧은 지연 시간 및 추론 프로세스 최적화와 같은 다른 QoS 특성을 달성하기 위해서는 여전히 DDNN 배포를 자동화해야 합니다.

Kafka-ML은 아파치 스톰, 아파치 스파크 스트리밍, 아파치 플링크[24]와 같은 다른 분산형 데이터 스트림 프레임워크와는 다른 접근 방식을 따릅니다. 이러한 프레임워크는 모든 규모의 데이터 스트림에서 분산 연산을 수행할 수 있도록 지원합니다. 아파치 사모아(Apache SAMOA)[25]는 아파치 스톰이나 아파치 삼자와 같은 기본 처리 엔진의 복잡성을 직접적으로 다루지 않고 데이터 스트림을 통해 ML 알고리즘을 개발할 수 있도록 하는 것을 목표로 하는 또 다른 프로젝트입니다. 일반적으로 이러한 프레임워크는 일반적으로 데이터 스트림으로 연산을 분산하기 위한 분산 엔진을 제공하지만, Kafka-ML처럼 널리 사용되는 ML/AI 프레임워크(예: 텐서플로우) 및 ML/AI 파이프라인을 촉진하는 데 특별한 초점을 두지 않거나 지원이 제한적입니다.

티라피타야논 등[26]은 BranchyNet 접근법을 통해 조기 종료 개념을 도입했습니다. 이 개념은 추론이 전체 심층 신경망 모델을 거치지 않고 중간 계층에서 끝날 수 있도록 합니다. 조기 종료를 사용하면 최종 정확도가 떨어질 수 있지만 응답 시간이 크게 단축된다는 점을 고려하면 수용할 수 있습니다. 이러한 조기 종료를 기반으로 심층 신경망을 분할하면 클라우드-사물 간 연속체에서 리소스 소비를 상당히 줄일 수 있으므로 이 개념을 따르는 DDNN은 연속체의 최상위 레벨로 메시지를 전송하는 것을 줄임으로써 시간에 민감한 애플리케이션의 응답 시간을 줄이는 데 더 큰 영향을 미칩니다.

또한 클라우드와 엣지 환경을 결합하면 응답 시간은 빨라지고 네트워크 혼잡은 줄어든다는 연구 결과도 있습니다 [27]. 그럼에도 불구하고 포그는 엣지 및 클라우드와 더불어 중요한 역할을 합니다. 이로 인해 포그 네트워크에서 동적 DDNN 파티셔닝을 위한 매칭 이론에 기반한 세분화된 솔루션인 DINA[28]가 탄생했습니다. 그럼에도 불구하고 BranchyNet [26]이 제안하는 바와 같이 중간 계층에서 추론이 조기에 중단되면 응답 시간뿐만 아니라 네트워크 트래픽 [29] 및 컴퓨팅 용량[30]도 감소하므로 조기 종료도 고려해야 합니다.

DIANNE [31]은 딥러닝 애플리케이션을 위한 유연하고 모듈화된 프레임워크를 제공하여 DNN을 주요 연산(예: 소프트맥스 출력 및 숨겨진 레이어)으로 분할하고 이를 여러 이기종 디바이스에 분산할 수 있는 서비스로 구현합니다. 이러한 접근 방식은 모듈성이 뛰어나지만 자체 개발 구성 요소로 제한되고 널리 사용되는 프레임워크가 아니기 때문에 솔루션 채택이 제한될 수 있습니다. 또한 DIANNE은 계층 간 통신을 최적화하기 위해 Branchynet의 조기 퇴출을 고려하지 않습니다.

적응형 수술 방식[32]은 DDNN을 동적으로 분할하여 엣지와 클라우드 사이의 다양한 네트워크 조건에서 지연 시간과 처리량을 모두 최적화합니다. 이 접근 방식은 추론이 중간 계층에서 중단되는 경우(조기 종료)를 고려하지 않습니다. 33]에서는 브랜치넷 네트워크의 추론 시간을 최소화하는 최적의 파티션을 찾기 위한 최적화 알고리즘을 제안합니다. AAIoT [30]에서는 다계층 IoT 시스템에서 추론 작업의 연산 할당을 최적화하기 위한 신경망 분할 방법을 제안합니다. Edgent [34]는 엣지 컴퓨팅에서 동적 DNN 협업 추론을 위한 프레임워크를 제시하며, 추론 정확도를 극대화하고 지연 응답을 최적화하기 위해 DDNN 동적 분할과 브랜치넷 조기 출구를 결합합니다.

이전 접근 방식의 주요 단점은 다음과 같습니다: 1) 카프카-ML과 같은 DDNN의 매니지먼트를 위한 오픈 소스 구현을 제공하지 않으며, 2) 결과의 더 나은 추정을 위해 텐서플로우와 같이 현재 사용되는 ML 프레임워크를 고려하지 않으며, 3) 일부는 시뮬레이션 결과만 수행하며, 마지막으로 4) IoT와 같이 다양한 조건과 데이터 스트림이 입력되는 인프라의 내결함성이나 부하 분산을 고려하지 않는다는 점입니다.

심층 신경망 모델은 수많은 숨겨진 레이어로 구성됩니다. 이를 분산 배치[21]하면 응답 시간이 향상되는 경향이 있으며[27], 특히 실시간 응답이 필요한 매우 복잡하고 중요한 문제에서 더욱 그렇습니다. 더 작은 신경망은 응답 지연을 크게 줄이고 리소스 제약이 있는 디바이스에 적합할 수 있습니다. 하지만 정확도가 떨어질 수 있으며, 따라서 예측에 더 높은 신뢰도가 필요할 수 있는 중요한 시스템의 신뢰도가 떨어질 수 있습니다. 이러한 의미에서 딥 뉴럴 네트워크의 파티셔닝에는 정확도, 지연 시간, 디바이스 요구 사항 간의 트레이드오프가 존재할 수 있습니다. 따라서 이전에 Kafka-ML에서 구상했던 것처럼 환경으로부터 정보를 수집하여 클라우드로 전송하는 IoT 디바이스와 애플리케이션이 모든 지연 시간과 대역폭 소비 문제를 안고 정보를 처리하는 대신, 이 작업에서는 엣지 및 포그 컴퓨팅과 같은 계층과 패러다임을 포함하여 IoT에서 클라우드에 이르는 연속체에서 할당할 수 있는 분산 모델을 구상하고 있습니다. 이러한 계층에서는 예측이 충분히 정확할 때 정보가 생성되는 곳(IoT 디바이스)에 최대한 근접하여 시간에 민감한 응답을 제공하기 위해 DDNN이 파견됩니다.

연속체에서 엣지 및 포그 레이어를 할당하면 지연 시간 및 대역폭 소비 문제를 개선하고 워크로드 분산으로 응답 시간을 단축할 수 있지만, 이러한 레이어를 사용하여 분당 수백 번의 계산이 어려운 추론을 실행하는 경우 과부하가 발생할 수 있습니다. 이러한 과부하는 조기 종료라고 불리는 것을 고려하는 BranchyNet 기반의 분할 모델을 사용하여 처리할 수 있습니다. 따라서 대규모 모델은 모델의 마지막이 아닌 초기 레이어에서 정확한 답을 반환할 수 있습니다. 또한 조기 출구를 통해 클라우드-사물 연속체에서 이러한 모델을 여러 아키텍처 레이어에 분할하여 분산할 수 있습니다. 결과적으로 중간 에지 및 포그 레이어는 모델의 작은 부분으로부터 추론할 수 있으며, 예측이 충분히 정확하다면 클라우드 응답을 기다릴 필요 없이 응답을 반환할 수 있습니다.

반면에 클라우드-사물 연속체에서 인프라를 관리하고 배포하는 데는 구성 요소의 동적 특성으로 인해 몇 가지 문제가 발생할 수 있습니다. 이러한 시나리오에서는 적절한 관리 및 내결함성 보장이 DDNN 배포에 필수적입니다. 또한 노드의 이동성으로 인해 DDNN은 큰 지연을 피하기 위해 쉽게 이식할 수 있어야 합니다. 이러한 의미에서 컨테이너화 기술은 Kafka-ML을 통해 입증된 바와 같이 DNN 애플리케이션의 배포에 적합한 접근 방식이며, 이는 연속체에서 제시된 다계층 인프라를 관리하고 DDNN을 배포하기 위해 해결해야 할 접근 방식입니다.

요약하자면, 우리는 Kafka-ML이 제공하는 기능을 확장하여 클라우드-사물 연속체에서 DDNN 및 조기 종료와 함께 작동할 수 있도록 하고, 분산 ML의 관리, 배포 및 공유를 지원하는 것을 목표로 하고 있습니다.

```
edge_input = keras.Input(shape=64,
    name='edge_input')
x = layers.Dense(64,
    activation=tf.nn.relu,
    name='relu')(edge_input)
output_to_cloud = layers.Dense(64,
    activation=tf.nn.relu,
    name='output_to_cloud')(x)
edge_output = layers.Dense(10,
    activation=tf.nn.softmax,
    name='edge_output')(output_to_cloud)
edge_model=keras.Model(inputs=[edge_input],
    outputs=[output_to_cloud, edge_output],
    name='edge_model')

cloud_input = keras.Input(shape=64,
    name='cloud_input')
x1 = layers.Dense(64,
    activation=tf.nn.relu,
    name='relu1')(cloud_input)
x2 = layers.Dense(128,
    activation=tf.nn.relu,
    name='relu2')(x1)
cloud_output = layers.Dense(10,
    activation=tf.nn.softmax,
    name='cloud_output')(x2)
cloud_model = keras.Model(inputs=
    cloud_input,
    outputs=[cloud_output],
    name='cloud_model')
```

목록 1. 카프카-ML을 위한 분산 ML 모델 예시.

모델, 메트릭 및 결과를 통해 고성능 및 고가용성 인프라를 구축할 수 있습니다.

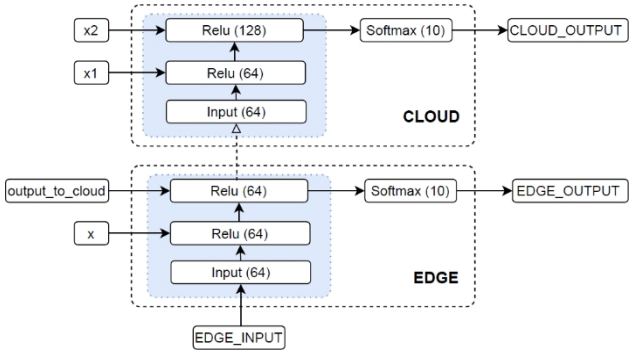
IV. KAFKA-ML의 분산 모델 정의

분산 모델이 무엇이고 어떻게 설계할 수 있는지에 대한 이해를 돕기 위해 분산 모델을 단계별로 정의하는 방법과 이를 ML 프레임워크인 텐서플로우에서 연결하기 위해 필요한 부분을 설명합니다.

목록 1과 2의 소스 코드가 익숙해 보일 수 있습니다. 실제로는 특정 숨겨진 레이어, 출력, 학습을 위한 컴파일이 포함된 간단한 Python TensorFlow/Keras 모델 정의입니다. 현재 Kafka-ML은 영역을 확장하기 위해 다른 ML 프레임워크와의 통합이 계속 개발 중입니다. 현재는 TensorFlow/Keras가 포팅되어 있습니다. 그림 2는 이 예제에 사용된 분산 심층 신경망 모델의 개요를 보여줍니다.

```
input = keras.Input(shape=64, name='input')
edge = edge_model(input)
cloud = cloud_model(edge[0])
model = keras.Model(inputs=[input],
 outputs=[edge[1], cloud], name='model')
model.compile(optimizer='adam',
    loss={
'edge_model':
    'sparse_categorical_crossentropy',
'cloud_model':
    'sparse_categorical_crossentropy'
    },
    metrics=['accuracy'],
    loss_weights=[0.001, 0.001])
```

목록 2. 전체 네트워크 정의 예시.



분산 모델을 만들 때 가장 먼저 고려해야 할 사항은 구현하고자 하는 글로벌 네트워크의 구조를 설계하는 것, 즉 네트워크의 일부가 될 분산 하위 모델을 정의하는 것입니다. 이 경우, 글로벌 모델이 엣지와 클라우드에 배포될 두 개의 파티션인 두 개의 서로 다른 하위 모델로 구성된 예가 목록 1에 나와 있습니다. 아키텍처가 완성되면

그림 2. 분산 신경망의 예시.

가 결정되면 다른 ML 프레임워크와 마찬가지로 사용자는 분산된 각 하위 모델의 내부 구조를 어떻게 구성할지 지정해야 합니다. 이 구조는 입력 레이어, 숨겨진 레이어, 출력 레이어로 구성됩니다. 이 예제에서 하위 모델들은 이미지를 받아들이기 위해 모양 64의 동일한 입력 레이어(*edge_input* 및 *cloud_input)를* 가지고 있습니다. 따라서 엣지 모델은 64개의 뉴런으로 구성된 하나의 숨겨진 레이어(리소스가 제한된 디바이스를 대상으로 함)와 두 개의 출력 레이어(*output_to_cloud* 및 *edge_output*)로만 구성됩니다. 이는 각각 클라우드에 대한 연결 레이어와 엣지에서의 초기 출구를 나타냅니다. 이러한 레이어는 글로벌 엣지 모델의 구성에서 출력으로 지정해야 합니다. 클라우드 모델은 64개 및 128개의 뉴런이 있는 두 개의 숨겨진 레이어(*x1* 및 *x2* 레이어)와 하나의 출력 레이어(*cloud_output*)로 구성됩니다. 이 경우, 이 연속체의 마지막 레이어이므로 조기 종료가 필요하지 않습니다. 이러한 분산 모델의 주목할 만한 측면은 정보가 글로벌 네트워크 전체에서 처리되기 때문에 부분적인 출력(예측)을 얻을 수 있다는 점입니다. 이는 연속체 인프라의 각 계층(엣지 및 클라우드) 내에 다양한 유형의 출력을 설정함으로써 달성됩니다. 예를 들어, 엣지 모델의 *소프트맥스* 출력 레이어 edge_output은 엣지에서 중간 예측을 생성하고 *cloud_output은* 클라우드에서 최종 출력을 생성합니다. 또 다른 중요한 출력은 다음을 담당하는 출력입니다.

분산 모델의 다음 계층으로 정보를 전송하여 계속 처리할 수 있도록 합니다(여기서 분산된 하위 모델의 결합이 수행됩니다). 이 경우, 엣지 모델의 ReLU 출력 레이어 *output_to_cloud*가 이 작업을 수행하며, 이는 클라우드 모델의 입력 레이어(*cloud_input*)에 연결됩니다. 따라서 output_to_cloud와 *cloud_input* 레이어는 동일한 차원을 가져야 합니다.

마지막으로, 훈련을 위해 목록 2와 같이 이전에 정의한 각 분산 하위 모델을 포함하는 전역 신경망을 만들어야 합니다. 지금까지 하위 모델에 대해 수행한 것처럼 전역 모델의 입력과 출력을 지정해야 합니다. 그러나 여기서는 각 분산 하위 모델을 정의하는 데 사용된 것과 동일한 구조(즉, 각 모델에 대해 출력이 정의된 순서)에 따라 상관 관계 모델의 출력과 입력을 수동으로 결합해야 하므로 특별한 고려 사항이 있습니다. 이 예에서 출력은 엣지의 초기 출구(*edge[1]*: *edge_output*)와 클라우드 출력입니다. 엣지 입력에 해당하는 입력은 하나뿐입니다. 이 과정은 학습 중에 Kafka-ML에 의해 자동으로 수행됩니다. 또한 이전 하위 모델의 출력 텐서 모양이 후속 하위 모델의 입력 텐서 모양과 동일한지 확인해야 합니다. 이 경우, 엣지 모델의 출력 레이어 *output_to_cloud*의 모양은 클라우드 모델의 입력 레이어 cloud_input의 모양(64 뉴런)과 같아야 합니다. 마지막으로 최적화기, 손실 유형, 원하는 메트릭 등 다양할 수 있는 일련의 파라미터를 설정하여 글로벌 모델을 컴파일해야 합니다. 서로 다른 하위 모델을 가진 글로벌 신경망을 정의할 때 한 가지 중요한 점은 분산된 하위 모델 각각에 대해 하나의 손실 유형을 지정하고 보유한 하위 모델 수와 동일한 수의 *손실 가중치*를 지정해야 한다는 것입니다. 이 예제는 다른 수의 분산 모델과 조기 종료로 외삽할 수 있습니다.

V. 카프카-ML의 분산 ML 모델 파이프라인

이 섹션에서는 다음과 같은 분산 ML 모델의 파이프라인을 설명합니다.
카프카-ML의 라이프사이클을 포함하여 소개합니다. 그 전에 카프카-ML의 중요한 개념인 컨피규레이션에 대해 소개하겠습니다. 구성은 훈련 및 평가를 위해 그룹화할 수 있는 ML 모델의 논리적 집합입니다. 이는 ML 모델 세트의 메트릭(예: 정확도 및 손실)을 비교 및 평가하거나 동일하고 *고유한* 데이터 스트림으로 병렬로 학습할 수 있는 모델 그룹을 정의해야 할 때 유용할 수 있습니다. 구성의 ML 모델은 동일한 데이터로 학습되므로 동일한 입력 레이어와 같은 동일한 유사성을 가져야 합니다. 한 구성에 하나의 ML 모델만 포함할 수도 있습니다.

Kafka-ML 파이프라인의 단계는 다음과 같습니다:
1) 분산형 ML 모델의 구현 및 등록
2) 학습할 ML 모델 세트에 대한 학습 구성 생성, 3) 특정 매개변수를 사용한 특정 학습 구성 배포, 4) 배포된 학습 구성에 피드 제공

구성, 5) 트레이닝 단계에서 결과 메트릭을 얻고 비교합니다;

6) 추론 및 사전 딕션 제작을 위해 선택한 ML 모델을 배포하고, 마지막으로 7) 배포된 학습된 모델에 데이터 스트림을 공급하여 예측 결과를 얻기 위한 추론 단계입니다. 그림 3은 파이프라인 단계의 전체 순서를 보여줍니다. 다음은 각 단계를 자세히 설명합니다.

그림 3. 분산형 카프카-ML의 ML/AI 파이프라인.

A. 분산형 ML 모델의 구현 및 등록

처음부터 저희는 프로그래머가 가장 중요한 일, 즉 ML 모델 생성에 집중할 수 있도록 Kafka-ML을 최대한 단순하게 만들고자 했습니다. 이것이 바로 ML 모델을 쉽게 테스트하고 검증할 수 있는 유용한 도구를 개발한 이유입니다. 따라서 필요한 소스 코드는 목록 1에 표시된 것처럼 분산된 ML 모델 정의뿐입니다. 반면에 목록 2에 표시된 것처럼 전체 네트워크 정의를 지정할 필요는 없습니다. 이 작업은 학습 단계에서 Kafka-ML이 자동으로 수행합니다.

전체 모델이 정의되면 그림 4와 같이 하위 모델을 분할하여 Kafka-ML 웹 UI에 삽입하여 모델을 생성할 수 있습니다. Kafka-ML에서 직접 모델을 정의할 수도 있지만, 다음과 같은 방법을 사용하는 것이 좋습니다.

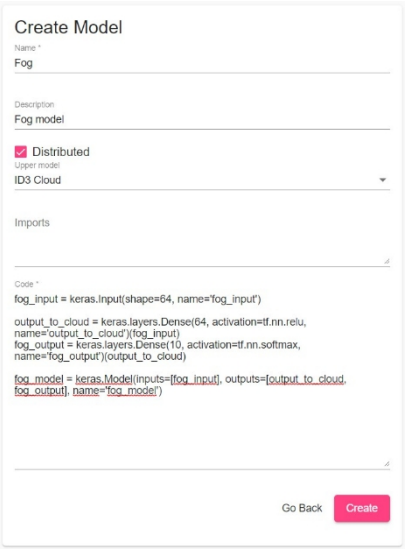


그림 4. Kafka-ML의 분산 ML 모델 정의.

보다 강력한 외부 ML 통합 개발 환경(IDE) 또는 Jupyter와 같은 편집기를 사용할 수 있습니다. 모델에 다른 필수 함수를 추가해야 하 는 경우 *가져오기* 필드에 삽입할 수 있습니다. 이 양식에는 배포와 관련된 두 개의 필드도 있습니다. 첫 번째 필드는 배포 모델을 생성하고 있는지 여부를 나타내고, 두 번째 필드는 이 모델에 연결된 다음 ML 배포 모델을 지정합니다. 모델이 제출되면 해당 소스 코드가 유효한 모델인지 확인하여 Kafka-ML에 통합합니다. 모델이 성공적으로 생성되면 파이프라인은 다음 단계로 넘어갈 수 있습니다.

B. 교육 구성 생성

구성은 단순히 나중에 함께 배포하고 학습할 ML 모델의 집합입니다. 전체 분산 모델을 추가하려면 구조의 맨 위에 있는 하위 모델만 선택하면 나머지 분산 체인은 자동으로 구성에 추가됩니다. 구성은 그림 5와 같이 Kafka-ML 웹 UI에서 생성할 수 있습니다.

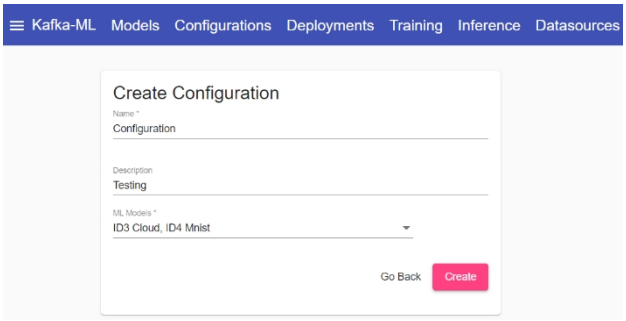


그림 5. Kafka-ML에서 구성 생성.

C. 교육용 구성 배포

구성을 배포하려면 먼저 배치 크기, 에포크, 반복 횟수 등 훈련 및 평가 단계와 관련된 몇 가지 매개 변수를 설정해야 합니다. 그런 다음 사용자가 이를 제출하면 Kafka-ML 모델별(전역) 작업이 배포됩니다. 모든 하위 모델은 동일한 작업에서 학습된다는 점에 유의하세요. 배포된 각 작업이 수행하는 첫 번째 단계는 Kafka-ML 기존 모델에서 해당 ML 모델(및 하위 모델)을 가져와서 로드하여 학습을 시작하는 것입니다. 결국, 작업은 훈련 데이터와 선택적으로 평가 데이터가 포함된 데이터 스트림이 Apache Kafka를 통해 수신될 때까지 기다립니다. 이렇게 하면 바로 학습할 수 있는 모델을 확보할 수 있고, Kafka에서 이미 사용 가능한 데이터 스트림이 있는 경우 바로 학습할 수 있습니다. 배포는 그림 6과 같이 Kafka-ML 웹 UI에서 생성할 수 있습니다.

D. 배포에 데이터 스트림 공급

파이프라인의 다음 단계는 훈련을 위해 데이터 스트림을 배포로 전송하는 것입니다. 훈련 단계는 Kafka 스트림 커넥터가 처음에 데이터 스트림이 있을 것으로 예상하는 대로 Kafka에서 데이터 스트림을 사용할 수 있을 때까지 시작되지 않습니다. 2

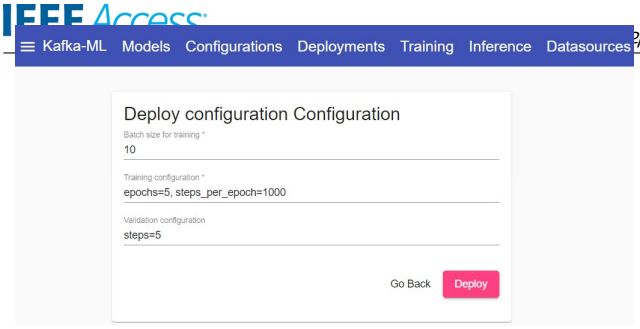


그림 6. Kafka-ML에서 학습을 위한 구성 배포.

첫 번째는 훈련 및 평가 단계에 대한 훈련 및 평가 데이터 스트림만 포함하는 데이터 토픽이고, 두 번째는 제어 메시지를 통해 배포된 ML 모델을 지정하고 *언제 어디서* 데이터 스트림을 훈련 및 평가에 사용할 수 있는지(데이터 토픽)를 지정하는 데 사용되는 제어 토픽입니다. 제어 메시지에 대한 자세한 내용은 Kafka-ML [15]에서 확인할 수 있습니다.

Kafka-ML에서 제공하는 라이브러리(AVRO 및 RAW)가 포함된 데이터 스트림(예: IoT 디바이스에서)을 해당 구성 배포로 전송하면, 구성에 포함된 모든 ML 모델이 학습을 시작하게 됩니다.

E. 교육 결과 메트릭 획득

학습 및 평가가 수행된 직후, Kafka-ML 사용자는 그림 7과 같이 학습된 각 모델(및 하위 모델)에 대해 정의된 메트릭(예: 손실 및 정확도)을 Kafka-ML 웹 UI에서 시각화할 수 있습니다. 이 양식에는 각 학습 작업이 Kafka-ML 아키텍처에 제출한 결과가 표시됩니다. 사용자는 각 모델에 대해 학습된 모델을 다운로드하거나, 삭제하거나, 추론(다음 단계)을 위해 배포할 수 있습니다.

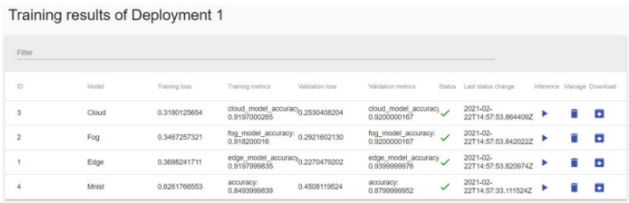


그림 7. Kafka-ML의 교육 관리 및 시각화.

F. 추론을 위한 ML 모델 배포

ML 모델이 학습되면 그림 8과 같이 Kafka-ML 웹 UI를 사용하여 추론을 위해 배포할 수 있습니다. 이 양식은 배포할 추론 복제본의 수를 묻습니다. 복제본은 추론 배포 간에 로드 밸런싱과 내결함성을 가능하게 합니다. 사용자는 양식에서 몇 가지 Kafka 토픽을 지정해야 합니다: 1) 예측할 값에 대한 입력 토픽, 2) 예측에 대한 출력 토픽, 그리고 분산형 ML 모델로 작업하는 경우, 3) 상위 토픽(상위 모델로 출력

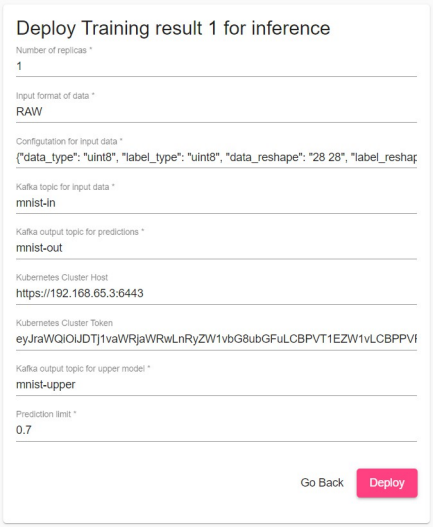


그림 8. Kafka-ML에서 추론을 위한 분산 훈련된 ML 모델 배포.

는 생산자(데이터 소스)와 수집자(학습 및 추론 작업)가 Apache Kafka와 상호 작용할 수 있는 기본적인 방법입니다. 각 주제는 이름으로 고유하게 식별됩니다. 모델(또는 하위 모델)을 컨티뉴엄의 다른 쿠버네티스 클러스터에 배포하기 위해 쿠버네티스 자격 증명을 포함할 수도 있습니다. 사용자는 또한 예측 한도를 정의해야 합니다. 이 한도는 하위 모델의 정확도 수준을 제어하는 데 사용되며, 이를 통해 예측을 다음 계층으로 올릴지 여부를 결정합니다. 따라서 정확도가 임계값보다 높으면 예측 흐름을 완료하는 사용 가능한 조기 종료로 출력을 보낼 수 있습니다.

G. 배포된 학습된 모델에 추론을 위한 데이터 스트림 제공

마지막으로, 학습된 모델이 준비되고 데이터 스트림을 통해 예측을 수행할 수 있도록 배포되면 ML/AI 파이프라인이 종료됩니다. 이 경우 추론에 필요한 정보(예: 입력 및 출력 토픽)가 이미 Kafka-ML 웹 UI에 정의되어 있기 때문에 제어 메시지를 전송할 필요가 없습니다(그림 8). 사용자와 시스템은 선택한 데이터 형식의 데이터 스트림을 입력 토픽으로 전송하기만 하면 되고, 모델이 예측을 수행하면 추론 결과가 즉시 출력 토픽으로 전송됩니다.

VI. 분산형 ML 모델을 위한 카프카-ML 아키텍처

Kafka-ML은 새로운 오픈 소스 프레임워크입니다. 데이터 스트림을 통해 ML/AI 파이프라인을 에이징합니다. 이 작업에서는 클라우드-사물 연속체에서 분산된 DNN 모델을 관리하고 배포할 수 있도록 Kafka-ML 아키텍처를 확장했습니다. 이전 섹션에서 살펴본 바와 같이, Kafka-ML은 ML/AI 전문가와 비전문가 모두 쉽게 따라할 수 있는 오픈 소스 웹 사용자 인터페이스(UI)를 제공하여 ML/AI 파이프라인을 처리할 수 있습니다. 사용자는 몇 줄의 ML 모델 코드만 작성하면 알고리즘을 학습, 평가, 평가하고 추론을 수행할 수 있습니다.

또한, 이 구조는 재사용이 가능한 Apache Kafka의 데이터 스트림을 처리하는 새로운 방법을 활용하므로 Kafka-ML의 데이터 세트에 대한 정보 시스템이나 데이터 스토리지가 필요하지 않습니다(이에 대해서는 섹션 VI-E에서 자세히 설명합니다).

Kafka-ML 아키텍처는 마이크로서비스 아키텍처를 구성하는 단일 책임 원칙에 기반한 일련의 구성 요소를 포함합니다[35]. 이러한 모든 구성 요소는 컨테이너화되어 Docker 컨테이너로 실행되므로 구성 요소를 격리하고 이식할 수 있습니다. 노드 클러스터의 관리와 분산 및 프로덕션 인프라에 Kafka-ML을 배포하는 것은 Kubernetes를 통해 이루어집니다. Kubernetes는 컨테이너와 그 복제본을 지속적으로 모니터링하여 컨테이너가 정의된 상태와 지속적으로 일치하는지 확인하고 고가용성 및 로드 밸런싱과 같은 프로덕션 환경을 위한 다른 기능을 사용할 수 있게 해줍니다. Kubernetes는 Kafka-ML과 그 구성 요소의 수명 주기를 관리합니다. 분산 머신 러닝 모델을 위한 Kafka-ML은 오픈 소스 프로젝트이며, 구현, 구성, Kubernetes 배포 파일 및 몇 가지 예제는 GitHub 리포지토리에서 확인할 수 있습니다.[2] 그림 9에는 Kafka-ML 아키텍처, 새로운 구성 요소, 이전 버전의 Kafka-ML과 관련하여 수정된 구성 요소에 대한 개요가 나와 있습니다. 이 개요에서 Kafka-ML은 클라우드의 Kubernetes 클러스터에 배포되고 하위 모델은 에지, 포그, 클라우드의 클러스터에 배포됩니다. 이 아키텍처에서 새로 생성된 구성 요소는 파란색으로 표시되고 수정된 구성 요소는 주황색으로 표시됩니다. 제어 로거는 제어 메시지를 관리하고 백엔드로 전송하는 역할을 담당합니다. Apache Kafka와 Zookeeper(Kafka에 필요)도 Docker 컨테이너로 배포되고 Kubernetes를 통해 관리됩니다.

A. 백엔드

백엔드 구성 요소는 Kafka-ML에 포함된 모든 정보를 관리하기 위한 RESTful API를 제공합니다. 이 구성 요소는 분산 파이프라인에 필요한 모든 구성 요소를 관리하고 배포하기 위해 Kubernetes API와 함께 작동합니다. 이 컴포넌트는 Python 웹 프레임워크인 Django와 Kubernetes용 공식 Python 클라이언트 라이브러리를 사용하여 구축되었습니다.[3] 를 사용하여 구축되었으며 Kubernetes 컴포넌트의 배포 및 관리에 사용됩니다. 이전 버전의 Kafka-ML과의 차이점 중에는 분산 신경망 작업에 필요한 새로운 기능(예: 모델 및 구성 정의)과 분산 모델의 훈련 및 추론 단계를 담당하는 새로운 구성 요소를 통합하는 모듈 자체의 변경 사항이 있습니다.

B. FRONTEND

프런트엔드 구성 요소는 사용자가 다음에서 모든 정보와 기능을 관리할 수 있는 웹 사용자 인터페이스를 제공합니다.

[2] https://github.com/ertis-research/kafka-ml
[3] https://github.com/kubernetes-client/python

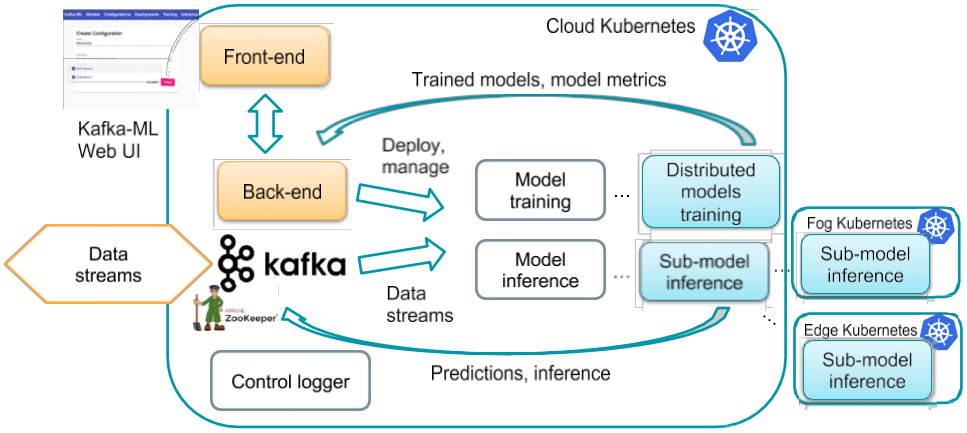


그림 9. 새로운 Kafka-ML 아키텍처와 Kubernetes 클러스터에서 배포 및 관리되는 해당 구성 요소의 개요. 주황색 구성 요소: 수정됨, 파란색 구성 요소: 추가됨.

Kafka-ML. 이 구성 요소는 백엔드에 정의된 RESTful API를 사용하며, 웹 개발용으로 널리 사용되는 TypeScript 프레임워크인 Angular를 사용하여 구현되었습니다. 이 경우, 분산 모델의 파이프라인을 완성하기 위해 각기 다른 양식에 필요한 모든 새로운 정보와 필드를 사용자에게 제공하기 위한 변경 사항이 포함되었습니다(예: 그림 4 및 8).

C. 분산 모델 학습

파티셔닝된 ML 모델이 포함된 특정 구성이 배포되면, 각 Kafka-ML 모델에 따라 잡(Kubernetes에서 배포 가능한 단위)이 실행되어 Docker 컨테이너에서 트레이닝 및 컨테이너화됩니다. 구성에 동일한 데이터 스트림으로 학습할 다른 분산 또는 비분산 모델이 포함되어 있는 경우, ML 모델당 작업도 생성됩니다. 알고리즘 1은 Kafka-ML에서 분산 학습 작업의 절차를 설명하며, 그림 10은 학습 과정의 시퀀스 다이어그램을 보여줍니다. 시퀀스 다이어그램에서 훈련 작업은 3개의 하위 모델로 구성된 모델에 대해 실행됩니다. 예외 관리 및 데이터 스트림 디코딩과 같은 일부 단계는 단순화를 위해 포함되지 않았습니다. 먼저 각 작업은 백엔드에서 분산 체인에 포함된 모든 ML 모델을 다운로드한 다음(1~7단계), 학습할 전체 분산 ML 모델을 구축합니다(8단계). 그런 다음, 작업은 예상한 메시지를 수신할 때까지, 즉 수신한 배포_id와 일치하는 메시지를 수신할 때까지 제어 스트림 메시지를 받기 시작합니다(12). 제어 메시지에는 데이터 스트림이 할당된 위치(13)와 훈련 및 평가를 위해 데이터 스트림이 지정된 방법도 표시됩니다. 훈련 및 평가는 제어 스트림 메시지에서 수신한 데이터 스트림 정보를 사용하여 수행됩니다. 훈련이 끝나면 작업은 분산 체인에서 훈련된 각 모델과 해당 훈련 및 평가 지표를 백엔드로 전송합니다(17~22). 대규모 연산 능력이 필요할 수 있는 이 프로세스는 클라우드와 같은 강력한 인프라의 Kafka-ML 인스턴스에서 수행할 수 있습니다.

알고리즘 1: 카프카의 분산 트레이닝 알고리즘 - 카프카의 분산형 트레이닝 알고리즘 ML

입력: models_urls, training_kwargs, evaluation_kwargs, deployment_id, 스트림 데이터

결과: 학습된 ML 모델 및 학습 및 평가 메트릭 배포

```
models[] ← downloadModelFromBackend(models_urls);
trained ← False;
훈련받지 않은 상태에서
    msg ← readControlStreams();
    if deployment_id == msg.deployment_id then training_stream ← readStream(msg.topic); if msg.validatition_rate > 0 then
        training_stream ← take(data_stream, msg.validation_rate);
        evaluation_stream ← split(data_stream, msg.validation_rate)
    끝
    model ← buildCompleteModel(models_urls);
    training_res ← trainModel(model, training_kwargs, training_stream);
    msg.validation_rate > 0이면 평가_res ← 평가모델(모델, 평가_kwargs, 평가_stream);
    끝 업로드트레이닝된모델및메트릭스 (models_urls, 모델, 훈련_res, 평가_res);
    훈련 ← 참입니다;
끝
끝
```

D. 분산 모델 추론

분산형 ML 모델이 학습되고 Kafka-ML 웹 UI를 통해 추론을 위해 배포되면, 복제 컨트롤러(복제 컨트롤러(복제 컨트롤러는 지정된

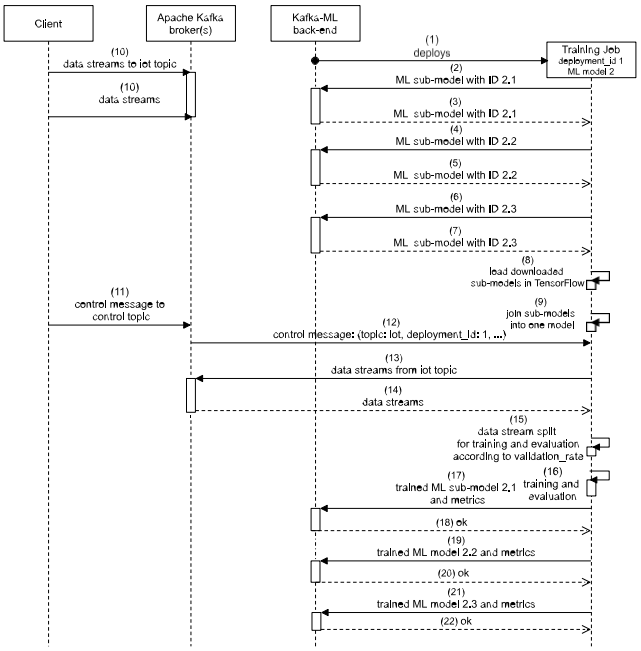


그림 10. Kafka-ML의 학습 프로세스 순서도.

알고리즘 2: 카프카의 분산 추론 알고리즘 - ML

입력: 모델 URL, 입력 주제, 출력 주제, 상위 주제, 입력 구성, 스트림 데이터, 제한

결과: 아파치 카프카 모델에 대한 예측 ←.

downloadTrainedModelFromBackend(model_url); 역직렬화기 ← getDeserializer(input_configuration); **while** *True* **do**
  stream ← readStreams(input_topic);
  data ← decode(역직렬화기, stream);
  predictions ← predict(model, data);
  if *max(predictions)* < *limit* **then**
    sendToKafka(예측, 상위_토픽);
  else
    sendToKafka(예측, 출력_토픽);
  끝
끝

항상 실행 중인 복제본 수)는 해당 Docker 컨테이너와 함께 설정된 복제본으로 실행됩니다. 복제본은 이 구성 요소의 로드 밸런싱과 고가용성을 보장합니다. 알고리즘 2는 Kafka-ML의 분산 추론 절차를, 그림 11은 시퀀스 다이어그램을 설명합니다. 시퀀스 다이어그램에서는 세 개의 하위 모델로 구성된 모델이 연속체의 세 계층에 배치되어 있습니다: 엣지, 포그, 클라우드입니다. 이 경우 분산 체인에서 ML 하위 모델별로 복제 컨트롤러가 실행되며, 이 모델이 배포될 Kubernetes 클러스터(예: 엣지, 포그, 클라우드)는 배포 중에 Kafka-ML에서 구성할 수 있습니다. 이 구성 요소가 요청된 학습된 하위 모델(4-5, 7-8, 10-11)의 다운로드를 완료하면 모델이 로드되고(6, 9, 12), 구성 요소는 데이터 스트림을 수신하기 시작하여 예측을 수행합니다(15). 그런 다음, 획득한 정확도가 배포 중에 설정한 임계값보다 높으면 설정한 Kafka 출력 토픽을 통해 예측 결과를 전송하고, 그렇지 않으면 부분적인 결과를 분산 체인의 다음 계층으로 전송하여 분산 흐름을 계속합니다(17 및 21). 이 예제에서는 데이터 스트림이 먼저 에지 레이어에서 수신되는데, 이 레이어는 충분한 정확도를 제공하지 못합니다. 그런 다음 데이터 스트림은 포그 계층으로 전송되어 관심 있는 애플리케이션이 클라우드에서 최종적으로 응답을 받습니다. 복제 컨트롤러는 복제본과 파티션을 일치시켜 로드 밸런싱과 더 높은 데이터 수집을 제공하기 때문에 여러 개의 Kafka 브로커와 파티션이 있는 경우 유용합니다.

E. 카프카-ML의 데이터 스트리밍 관리

Kafka-ML의 데이터 스트림 관리는 학습과 추론이라는 두 가지 주요 작업과 관련하여 제공됩니다. 카프카

토픽은 카프카-ML에서 데이터 소스와 카프카-ML 작업을 연결하는 기본 방법으로, 카프카-ML 최종 사용자가 전송한 데이터 스트림을 할당합니다. 토픽은 서로 독립적이며 정의된 이름으로 고유하게 식별됩니다. 전송된 정보의 가용성을 보장하기 위한 내결함성과 부하 분산은 토픽의 파티션에 의해 제공되며, 각 토픽은 여러 개의 파티션으로 나눌 수 있고 각 파티션에는 여러 개의 복제본이 있을 수 있습니다. 파티션을 사용하면 로그를 더 작은 단위로 분할하여 부하 분산 기능을 제공할 수 있으며, 토픽 복제본을 사용하면 복제를 통해 내결함성을 확보할 수 있습니다. 토픽은 아파치 카프카에서 데이터 스트림을 전송할 때 자동으로 생성되거나, 카프카 매니저와 같은 오픈 소스 도구를 사용하여 정의할 수 있습니다.[4] 마지막으로, 토픽은 카프카의 아키텍처를 구성하는 카프카 브로커 클러스터에 의해 관리되며, 데이터 소스에서 데이터 스트림을 수신하고 이를 구독하는 소비자에게 배포(예: 학습 및 추론 작업)하는 역할을 담당합니다. Kafka를 통해 전송되는 데이터 스트림은 Kafka-ML에서 지원되는 AVRO 및 RAW 데이터 형식을 사용하여 인코딩해야 합니다. 이를 위해 사용자가 간단한 방법으로 데이터 스트림과 제어 메시지를 코딩하여 전송할 수 있도록 프로젝트 저장소에 몇 가지 스크립트가 제공되어 있습니다.

학습을 위해 전송되는 데이터 스트림은 Apache Kafka에서 제공하는 분산 로그를 사용하며, 학습 작업은 로그를 따라 이동하고 제어 메시지에 표시된 대로 데이터 스트림을 읽습니다. 데이터 스트림을 전송한 직후 제어 메시지가 전송되며, 이 메시지에는 전송된 데이터 스트림이 정확히 어디(분산 로그의 주제 및 위치)에서 사용 가능한지 알려줍니다. 덕분에 전송된 동일한 데이터 스트림을 일부 제어 메시지만 보내면 다른 학습 작업에서 재사용할 수 있을 뿐만 아니라, 실패하여 전체 데이터 스트림을 복구해야 하는 학습 작업에도 사용할 수 있습니다. 각 메시지를 삭제할 수 있는 기존 메시지 큐 시스템에서

[4] https://github.com/yahoo/CMAK

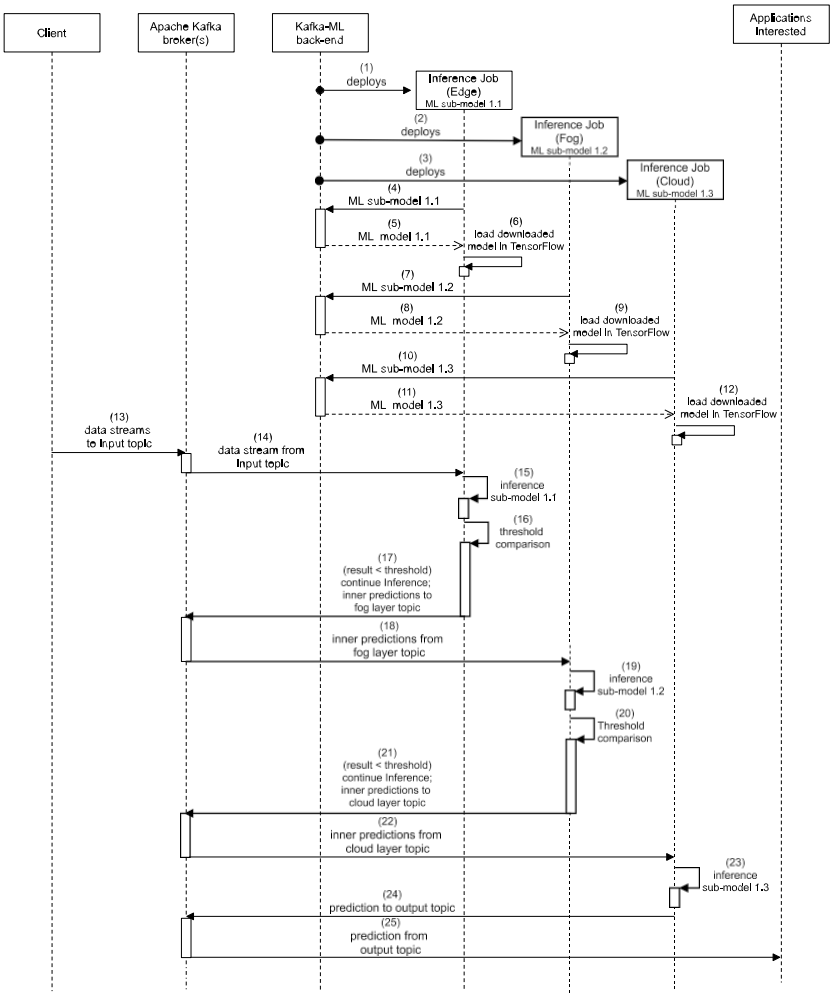


그림 11. Kafka-ML의 추론 프로세스 순서도.

사용 후에는 이러한 상황에서 데이터 손실이 발생하지 않도록 데이터스토어가 필요할 수 있습니다. 트레이닝을 위한 데이터 스트림 관리의 예는 그림 12에 나와 있습니다.

먼저, 녹색 데이터 스트림이 제어 메시지 C1과 함께 배포된 구성 D1로 전송되었습니다. 구성 D2가 동일한 데이터 스트림을 사용할 수 있도록 제어 메시지 C1이 다시 전송되었습니다. 현재 상태에서는 녹색 데이터 스트림이 만료되어 다른 학습 작업에 더 이상 재사용할 수 없습니다. 그런 다음 파란색 데이터 스트림은 구성 D3 및 D5에 대해 전송되고 주황색 데이터 스트림은 구성 D4에 대해 전송됩니다. 주황색과 파란색 데이터 스트림은 이 데이터 스트림을 사용하려는 새 구성에 계속 재사용할 수 있습니다. 마지막으로 회색 데이터 스트림이 이제 분산 로그에 입력되고 있으며, 데이터 스트림이 완료되지 않았기 때문에 제어 메시지가 아직 전송되지 않았습니다.

추론의 경우 제어 메시지를 전송할 필요가 없으므로 프로세스에 필요한 단계가 더 적습니다(데이터 스트림은 하나씩 처리할 수 있음). 데이터 스트림은 선택한 인코딩에 따라 데이터 소스(예: IoT)에 의해 입력 토픽으로 하나씩 전송되며, 추론 결과는 조기 종료 출력 토픽으로 발송됩니다(

정의된 임계값보다 높음) 또는 클라우드-사물 컨티뉴엄의 상위 계층으로 전송하여 추론 프로세스를 계속 진행합니다. Kafka-ML에서 동일한 입력 토픽을 구성하기만 하면 여러 추론 배포에서 동일한 데이터 스트림을 처리할 수 있습니다(그림 8). 이렇게 하려면 추론 ML 모델이 동일한 입력을 받아들여야 한다는 점에 유의하세요. 마지막으로, 더 많은 데이터 스트림과 데이터 소스를 지원하기 위해 추론 모듈은 추론 그룹을 구성하는 복제와 함께 배포할 수 있습니다. 이 경우, 그룹의 각 구성원은 한 번에 하나의 데이터 스트림을 수신하게 되며, Apache Kafka의 소비자 그룹 기능 덕분에 그룹 전체에 부하가 분산됩니다.

VII. 평가

앞서 설명한 바와 같이, 이 작업에서 제시된 Kafka-ML 확장을 통해 사용자는 분산 DNN을 관리하고 배포할 수 있습니다. Kafka-ML의 이 새로운 기능을 평가하기 위해 다양한 시나리오를 정의했습니다. 이러한 시나리오는 그림 1에 정의된 클라우드-사물 연속체에 따라 세 가지 가능한 경우를 포함합니다. 첫 번째 시나리오는 예측을 위해 클라우드를 독점적으로 사용하여 다음을 전송하는 경우입니다.

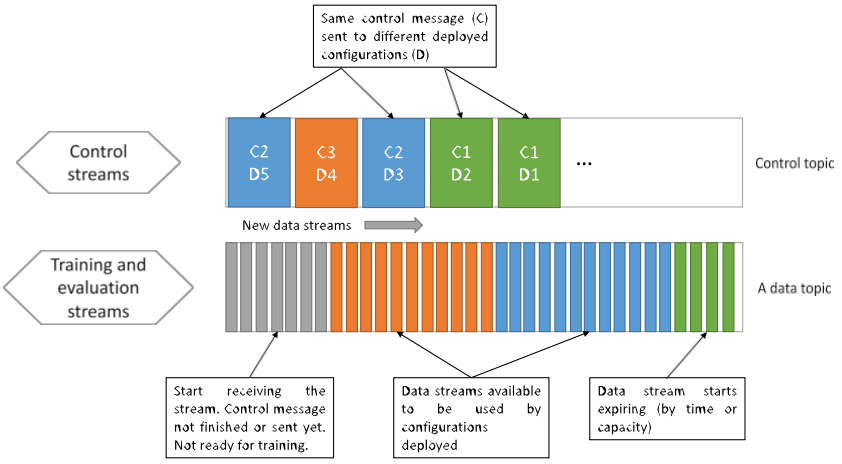


그림 12. Kafka-ML의 데이터 스트림 관리 [15].

모든 계산 부하를 이 레이어에 할당합니다. 다른 두 가지 시나리오에서는 각각 엣지 레이어와 엣지-포그 레이어를 고려합니다. 우리는 BranchyNet [26]과 AlexNet [36]을 기반으로 기본 ML 모델을 정의했습니다. AlexNet은 ImageNet LSVRC-2010 대회에서 최첨단 결과를 얻은 잘 알려진 심층 신경망입니다. 저희는 Kafka-ML을 사용하여 다양한 모델 버전의 훈련 및 평가 프로세스에 CIFAR10 [37] 데이터 세트를 사용했습니다. CIFAR10은 균형이 잘 잡힌 10개의 클래스로 구성된 6000개의 32×32 컬러 이미지로 구성되어 있습니다. 훈련 단계에서는 80%, 테스트 단계에서는 나머지 20%를 사용했습니다. 다양한 시나리오에 맞게 모델을 조정하기 위해 초기 출구를 고려했기 때문에 ML 모델 버전은 각 사례마다 약간씩 다릅니다:

1) **시나리오 1**. 이 시나리오에서는 IoT 디바이스가 클라우드에 직접 요청을 전송하는 경우를 고려합니다. 즉, 클라우드가 모든 요청(즉, 모델 추론)을 완전히 처리하며 에지 및 포그 디바이스가 없습니다. 따라서 조기 종료 기능을 포함하지 않았으며, 이 경우 모델을 여러 하위 모델로 분할하지 않았습니다. Kafka-ML에 ML 모델을 도입한 후 프레임워크는 이를 학습하고 클라우드에 구성된 Kubernetes 클러스터에 배포합니다. 그림 13은 클라우드에서 예측 프로세스를 위해 배포된 ML 모델을 보여줍니다.
2) **시나리오 2**. 그림 14는 그림 13에 표시된 모델을 엣지 레이어용 모델과 클라우드 레이어용 모델이라는 두 개의 하위 모델로 나눈 것을 보여줍니다. Kafka-ML에서 이러한 하위 모델을 구성한 후 사용자는 Kafka-ML에서 학습을 수행한 다음 클라우드-사물 간 계층의 여러 클러스터에 배포할 수 있습니다. 따라서 IoT 디바이스는 이 시나리오에 대한 예측을 시작하기 위해 엣지 클러스터에 요청을 전송합니다. Edge 클러스터는 사전 예측 프로세스의 일부를 처리하고 추가된 조기 종료로 결과를 제공합니다. Edge 조기 종료에 의해 수행된 예측이 충분히 정확하지 않은 경우(즉, 반환된 예측 확률이 지정된 임계값보다 작은 경우), Edge는 클라우드에 요청을 전송합니다.

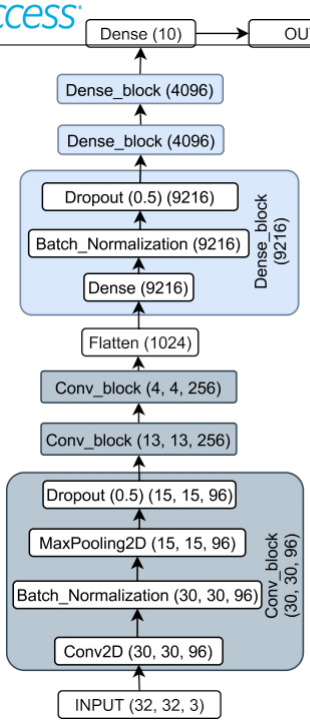


그림 13. 시나리오 1: 클라우드에 배포된 AlexNet 브랜치넷 기반 DDNN.

를 눌러 추론 프로세스를 계속 진행합니다. 클라우드-투-사물 *컨티뉴엄의* 상위 레벨로 정보를 전송할지 여부를 결정하기 위해 임계값을 0.8로 설정했습니다. 이 값보다 높은 초기 종료 응답의 대부분이 클라우드에서 반환된 CIFAR10의 클래스와 일치하기 때문에 이 값을 고려했습니다. 임계값을 높이면 더 높은 정확도를 제공할 수 있습니다. 반대로 이 값을 낮추면 엣지 계층이 클라우드 계층으로 보내는 요청이 줄어들기 때문에 응답 속도가 빨라집니다. 따라서 이 임계값은 정확도와 응답 시간 사이의 절충안입니다.

3) **시나리오 3**. 마지막으로, 마지막 시나리오에 추가 조기 종료 기능을 포함하고

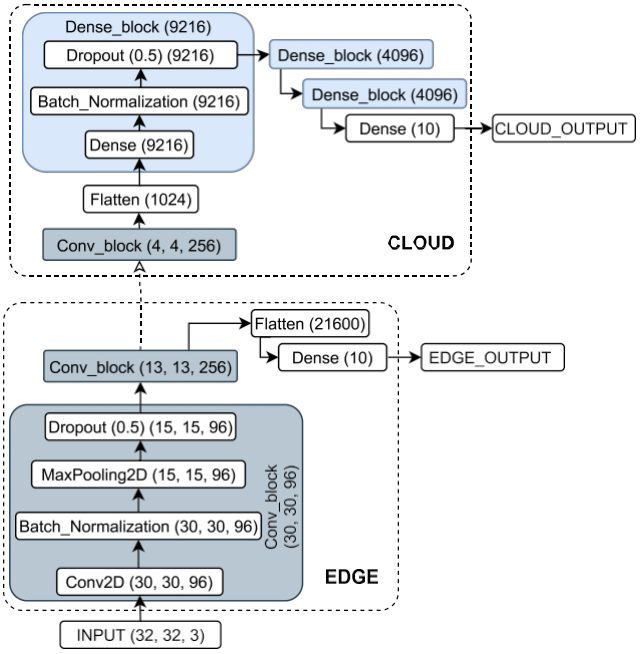


그림 14. 시나리오 2: 엣지 및 클라우드에 배포된 AlexNet 브랜치넷 기반 DDNN.

Edge 하위 모델을 두 개의 하위 모델로 나눕니다. 이러한 하위 모델 중 하나는 포그 레이어에 대한 신경망 레이어를 포함합니다. 나머지 신경망 레이어는 최종 엣지 하위 모델을 구성합니다. 따라서 이 시나리오에는 그림 14에서 고려한 엣지 및 클라우드 외에 추가 레이어가 연속체에 포함됩니다. 이 경우 Kafka-ML은 필요할 때 엣지, 포그, 클라우드 레이어를 연결하여 모델을 배포합니다. 앞의 경우와 마찬가지로 IoT 디바이스는 엣지 디바이스로 요청을 전송하며 *임계값은* 0.8입니다. 하지만 이 경우에는 클라우드 레이어가 예측 프로세스를 계속하는 대신 엣지 레이어가 포그 레이어와 통신합니다. 그런 다음, 포그 레이어는 포그 조기 종료에 의해 주어진 확률이 포그 레이어의 임계값보다 낮으면 클라우드 레이어에 예측을 종료하도록 요청합니다(엣지 레이어에서와 동일). 그림 15는 이 시나리오의 하위 모델과 각각의 조기 종료를 나타냅니다.

이 세 가지 시나리오에서는 네 가지 아키텍처 계층을 식별합니다. 모든 경우에서 각 계층에 동일한 리소스가 있습니다. 하지만 모든 시나리오에서 이러한 계층 중 일부를 사용하는 것은 아닙니다. 네 가지 계층의 구성은 다음과 같습니다:

- **IoT 또는 장치**: 정보를 전송하고 최종 결과를 측정하기 위해 한 대의 PC를 클라이언트로 구성했습니다. 이 컴퓨터에는 16GB RAM과 i5-7400 3.00GHz 프로세서가 장착된 Windows 10이 있습니다. 이 PC를 사용하여 점점 더 많은 클라이언트의 예측 요청을 시뮬레이션했습니다.
- **엣지**: 엣지 계층은 62GB RAM과 3.70GHz의 i9-10900K CPU가 장착된 단일 컴퓨터에 구축되어 있습니다. 이 컴퓨터에는 Kubernetes v1.21.2와 Docker가 있습니다.
  20.10.7은 우분투 21.04에서 실행 중입니다.

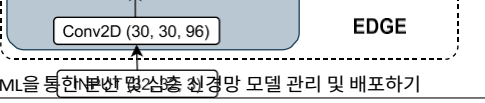


그림 15. 시나리오 3: 엣지, 포그, 클라우드에 배포된 AlexNet 브랜치넷 기반 DDNN.

- **포그**: 프라이빗 VMWare vCloud 인프라의 5노드 Kubernetes 클러스터는 포그 계층으로 구성됩니다. 클러스터는 마스터 노드와 4개의 워커로 구성됩니다. 이 클러스터의 각 노드에는 총 4개의 vCPU와 16GB RAM이 있습니다. 각 노드는 Ubuntu를 실행합니다. 16.04.7 LTS(Kubernetes v1.19.3 및 Docker 19.03.13 포함).
- **클라우드**: 클라우드 계층에 대한 구성은 Google Cloud Platform에서 설정되었습니다. 프레임워크의 새로운 기능을 평가하기 위해 6개의 노드, 12개의 vCPU, 24GB 메모리로 Kubernetes 클러스터를 구성했습니다.

이러한 시나리오와 위에서 언급한 구성을 고려하여 다양한 테스트를 수행했습니다. 이 테스트는 데이터 수집을 늘리고(사전 받아쓰기를 요청하는 클라이언트 수를 제어), 카프카 토픽의 복제 및 파티션을 증가시킴으로써 카프카-ML의 성능을 평가하는 데 목적이 있습니다. 처음에는 단일 Kafka 브로커를 사용하는 간단한 시나리오를 배포한 다음, 나중에 세 개의 Kafka 브로커를 사용하여 고가용성 기능을 평가했습니다. 최근의 실험은 새로운 Kafka-ML 기능의 내결함성 및 고가용성 특성을 평가하기 위한 것입니다. 명확성을 위해 각 시나리오에 대해 수행된 평가를 다음 사항에 나열했습니다:

- 하나의 카프카 브로커를 사용한 카프카-ML 추론 인스턴스의 데이터 수집 성능.
- 3개의 카프카 브로커를 사용하여 고가용성 시나리오에서 배포.
- Kafka-ML에서 분산 DNN의 내결함성 및 고가용성 특성을 평가합니다.

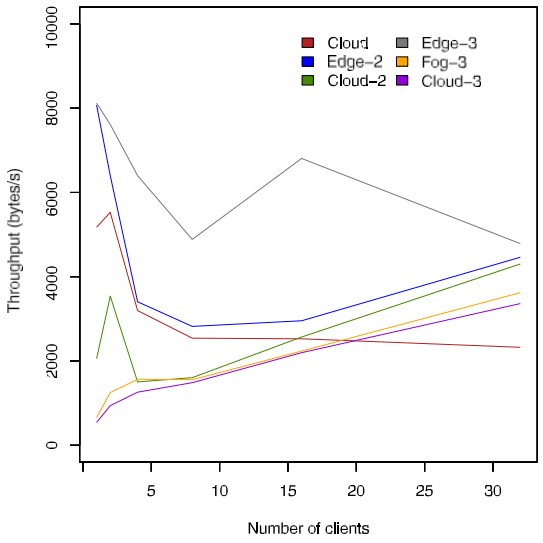
그림 16. 다양한 클라이언트 수 (복제본 1개, 카프카 브로커 1개)에 따른 평균 추론 지연 시간 응답.

모든 분석은 고객당 하나의 무작위 이미지를 사용하여 수행되었으며, 모든 고객이 동시에 예측을 요청했습니다. 이 섹션에 표시된 결과에 대한 통계적 신뢰도를 높이기 위해 각 사례에 대해 이 과정을 독립적으로 25회 반복한 후 평균 결과를 얻었습니다.

A. 테스트 1. 하나의 카프카 브로커를 사용한 데이터 수집

각 시나리오에 대해 수행되는 첫 번째 평가는 클라우드-사물 간 연속체(Cloud-to-Things Continuum)의 각 계층에 있는 단일 Kafka 브로커를 고려합니다. 주로 클라이언트 수의 증가가 추론 성능에 어떤 영향을 미치는지 분석하고자 합니다. 따라서 각 시나리오의 각 아키텍처 레이어에서 평균 지연 시간과 네트워크 통신 처리량을 측정했습니다. 평균 지연 시간은 발신자 디바이스(예: IoT 디바이스)가 사전 딕셔너리를 요청한 이후 평균 응답 시간을 의미합니다. 네트워크 통신 처리량은 클라우드-사물 간 연속체의 각 계층에서 초당 전송 및 처리되는 바이트 수를 나타냅니다.

모델 추론 서비스의 단일 복제본을 사용하여 가장 기본적인 배포에서 이 기능이 어떻게 작동하는지 평가할 수 있습니다. 그림 16은 하나의 카프카 브로커와 단일 복제본을 사용



하는 클라이언트 수가 증가할 때 추론 지연 시간 응답의 평균 값을 보여줍니다. 시나리오 1은 클라우드 출력, 시나리오 2는 엣지-2 및 클라우드-2 출력, 마지막으로 세 개의 레이어가 있는 시나리오 3은 각각 엣지-3, 포그-3, 클라우드-3에 해당합니다. 이는 다음 테스트에도 적용됩니다. 모든 시나리오에서 클라이언트 수가 증가함에 따라 응답 시간도 증가합니다. 클라우드에서 파티션되지 않은 모델을 나타내는 시나리오 1(클라우드)의 경우, 클라이언트 수가 적을 때 평균적으로 얻은 시간이 포그(Fog-3) 및 클라우드의 응답 시간보다 더 나은 것을 관찰할 수 있습니다.

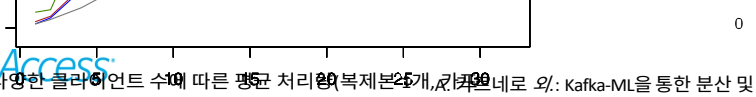
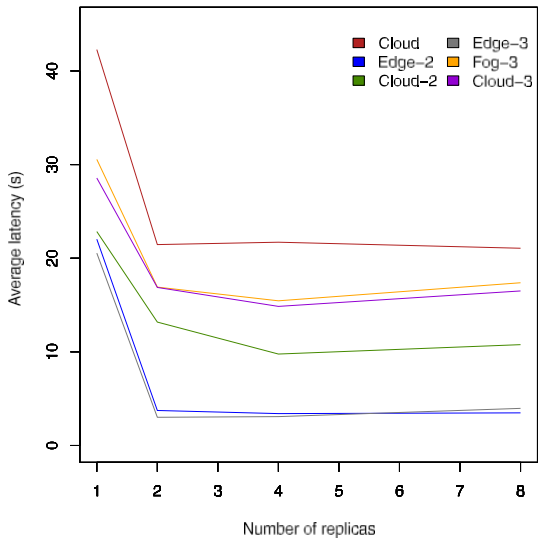


그림 17. 다양한 클라이언트 수에 따른 평균 처리량(복제본 2개, 카프카 브로커 1개).

(클라우드-2 및 클라우드-3) 레이어보다 훨씬 빠릅니다. 그럼에도 불구하고 나머지 시나리오(엣지-2 및 엣지-3)에서 엣지 레이어가 제공하는 응답 시간은 시나리오 1에 비해 현저히 낮습니다. 이러한 결과는 항상 요청을 클라우드 계층으로 보내는 대신 엣지 계층에서 많은 결과를 평가하고 반환하므로 응답이 더 빨라진다는 것을 의미합니다. 시나리오 2와 시나리오 3의 엣지 위 레이어 결과는 중간 레이어가 있기 때문에 이러한 레이어에 도달해야 하는 추론의 응답 시간이 엣지 레이어에 비해 증가한다는 것을 보여줍니다. 그러나 이러한 시나리오는 요청과 클라이언트 수가 많을 때 시나리오 1의 느린 응답을 상당히 보완합니다. 그림 16에서 볼 수 있듯이 클라이언트 수가 증가함에 따라 시나리오 1의 지연 시간이 크게 증가하기 시작하고 다른 두 시나리오 중 하나가 훨씬 더 효율적입니다. 시나리오 2와 시나리오 3에서 얻은 결과를 비교하면, 엣지의 컴퓨팅 부하 감소(시나리오 3 엣지 하위 모델의 레이어 수 감소)가 이 레이어의 응답 시간 및 부하를 줄이는 데 도움이 된다는 것을 알 수 있습니다. 클라이언트가 증가함에 따라 시나리오 1에서 지연 시간이 증가하는 것과 유사하게, 시나리오 2에서도 요청이 증가함에 따라 지연 응답이 증가합니다. 따라서 총 요청 수가 크게 증가하는 경우 연속체에서 세 번째 레이어(시나리오 3)를 사용하는 것이 더 유리합니다. 이 경우 엣지 레이어(Edge-3)는 더 낮은 응답 시간을 제공하지만 상위 레이어(Fog-3 및 Cloud-3)는 Cloud-2보다 높은 응답 시간을 제공합니다. 이 경우 시나리오 3에서 엣지가 제공하는 빠른 응답 시간(대부분의 경우 응답)과 상위 계층의 지연 시간 증가 사이에 절충점이 있습니다.

이러한 결과를 뒷받침하기 위해 다양한 클라이언트 수를 사용하여 평균 처리량을 살펴봤습니다. 그림 17은 클라이언트 수가 많은 시나리오 1이 초당 처리하는 바이트 수가 더 적다는 것을 보여줍니다. 반면 시나리오 i2와 시나리오 3은 초당 더 많은 정보를 처리할 수 있습니다.



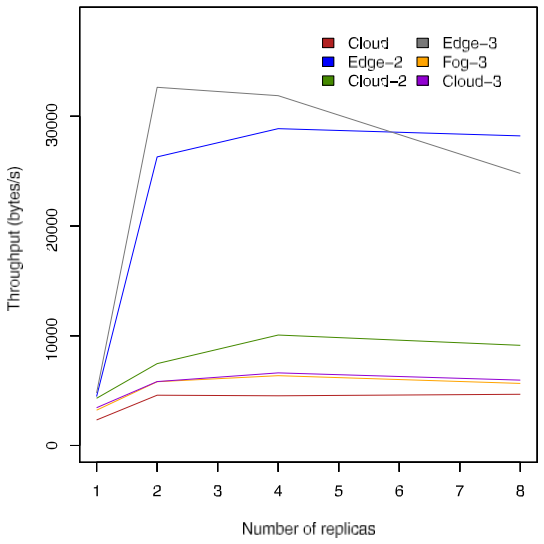


그림 18. 다양한 레플리카 수와 1개의 카프카 브로커에 대한 평균 추론 지연 시간 응답.

그림 17은 또한 첫 번째 레이어의 처리량이 가장 높다는 것을 보여줍니다. 즉, 엣지 레이어는 가능한 한 많은 정보를 처리할 수 있어야 합니다. 이렇게 하면 연속체의 상위 레이어는 과부하가 덜 걸리고 이러한 노력이 필요하지 않은 경우를 피하면서 훨씬 더 많은 비용이 드는 작업을 수행할 수 있습니다.

그림 18은 모든 레이어에서 추론 모듈의 복제본 수를 늘렸을 때 32개의 클라이언트가 있는 경우의 동작을 보여줍니다. 복제본의 수가 응답 시간에 큰 도움이 된다는 것을 알 수 있습니다. 그러나 두 개 이상의 복제본을 사용하면 안정화되는 경향이 있거나 심지어 더 나쁜 결과를 제공하는 것을 볼 수 있습니다. 이러한 동작은 하나의 브로커와 하나의 파티션으로 복제본 수를 늘리는 것만으로는 이러한 기능을 만족스럽게 활용하지 못하고 복제로 인해 시스템에 과부하가 걸리기 때문입니다. 그림 19를 보면 이 사실을 확인할 수 있습니다. 이 그림은 다양한 수의 복제본과 32개의 클라이언트에 대한 처리량을 나타냅니다. 두 개 이상의 복제본을 사용할 때 처리량은 지연 시간과 마찬가지로 큰 영향을 받지 않는다는 것을 알 수 있습니다.

B. 테스트 2. 3개의 카프카 브로커를 사용한 고가용성 환경

단일 카프카 브로커를 사용해 응답을 관리하면 레이어 간 통신에 병목 현상이 발생할 수 있습니다. 즉, 각 레이어에 있는 고유한 브로커가 모든 예측 메시지를 수신하고 전송해야 합니다. 따라서 레이어 간 중개자의 업무 과부하가 발생하게 됩니다. 이러한 이유로 이 테스트에서는 고성능 시나리오에서 Kafka-ML의 동작을 평가하기 위해 3개의 Kafka 브로커를 사용했습니다. 브로커 수를 늘림으로써 Kafka-ML에서 이러한 고성능 시나리오가 미치는 영향과 더 많은 요청으로 인한 병목 현상을 어떻게 줄일 수 있는지 평가할 수 있습니다.

그림 19. 다양한 레플리카 수와 1개의 카프카 브로커를 사용한 평균 처리량.

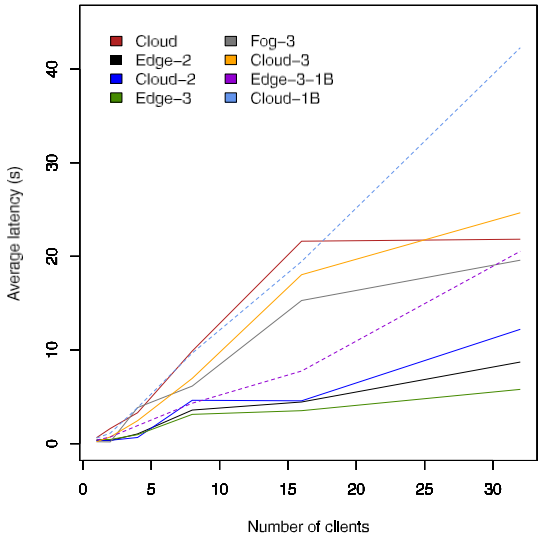


그림 20. 다양한 클라이언트 수 에 따른 평균 추론 지연 시간 응답(복제본 1개, 카프카 브로커 3개).

그림 20에는 클라이언트 수가 서로 다른 세 개의 카프카 브로커를 사용했을 때 얻은 평균 응답 시간이 표시되어 있습니다. 또한, 테스트 1에서 단일 카프카 브로커(각각 클라우드 및 엣지-3)를 사용한 최악의 결과와 최상의 결과를 가진 클라우드-사물 연속체 레이어를 나타내기 위해 여러 개의 불연속적인 선이 포함되었습니다. 시나리오 1에서는 클라우드가 여전히 대부분의 경우 더 나빴지만 다른 시나리오에서는 크게 개선되었습니다. 시나리오 2와 시나리오 3은 모든 레이어에서 응답 시간이 상당히 단축되어 시나리오 1보다 더 나은 시간을 기록하며 테스트 1에서 가장 좋은 결과를 얻었습니다(그림 16). 여기에서는 세 개의 Kafka 브로커를 사용한 이 배포에서 Kafka-ML의 향상된 성능을 보여줍니다.

이러한 결과는 그림 21을 통해 더욱 뒷받침됩니다. 이 그림은 다음과 같은 다양한 수에 대한 처리량을 고려한 것입니다.

클라이언트를 사용했을 때입니다. 이전 테스트에서와 마찬가지로 엣지 레이어가 첫 번째 시나리오에서 제안한 클라우드보다 훨씬 더 많은 정보를 처리하는 것을 볼 수 있습니다.

또한, 이전 테스트의 그림 21에서 점선을 보면 알 수 있듯이 3개의 Kafka 브로커를 사용하면 처리되는 정보가 훨씬 더 많아집니다. 엣지 수준에서 훨씬 빠른 응답을 제공함으로써 포그와 클라우드에 과부하가 걸리지 않고 컴퓨팅 용량이 필요한 요청에 집중할 수 있습니다.

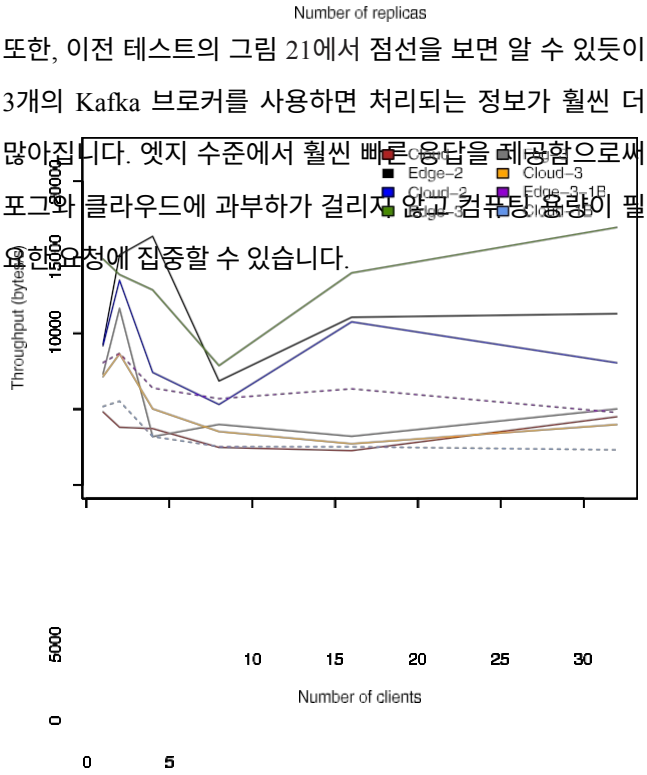


그림 21. 다양한 클라이언트 수에 따른 평균 추론 처리량(복제본 1개, 카프카 브로커 3개).

이전 테스트에서와 마찬가지로 복제본 수를 늘려도 크게 개선되지 않는 것으로 보입니다. 그러나 그림 22에서 볼 수 있듯이 클라이언트가 32개이고 복제본 수가 다른 경우 응답 시간은 각 계층에서 최소값에 도달합니다. 이 동작은 레플리카의 증가가 응답 시간에 도움이 된다는 것을 보장합니다. 따라서 요청과 클라이언트 수가 많을수록 더 큰 폭의 감소를 볼 수 있습니다. 그림 22는 레플리카 수가 다음과 같은 영향을 미치지 않음을 보여줍니다.

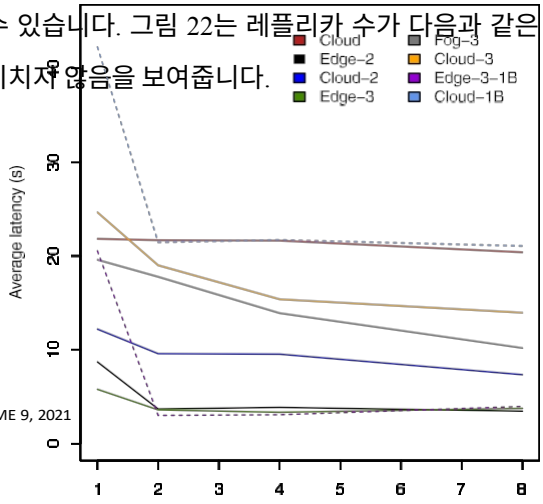


그림 22. 다양한 복제본 수와 3개의 카프카 브로커를 사용한 평균 추론 지연 시간 응답.

는 시스템의 과부하를 의미하며, 단일 복제본으로 그림 20에서 얻은 결과를 개선합니다. 복제본이 브로커를 공유하여 지연 시간을 줄일 수 있기 때문에 더 많은 수의 브로커가 있는 Kafka-ML을 배포하면 복제를 선호하는 데 기여합니다.

그림 23에서 두 개의 복제본을 사용할 때 대부분의 레이어에서 처리량이 크게 향상되는 것을 볼 수 있습니다. 또한 복제본이 증가하면 성능이 저하되었던 이전 테스트와 달리 위의 복제본 증가는 약간의 개선을 나타냅니다.

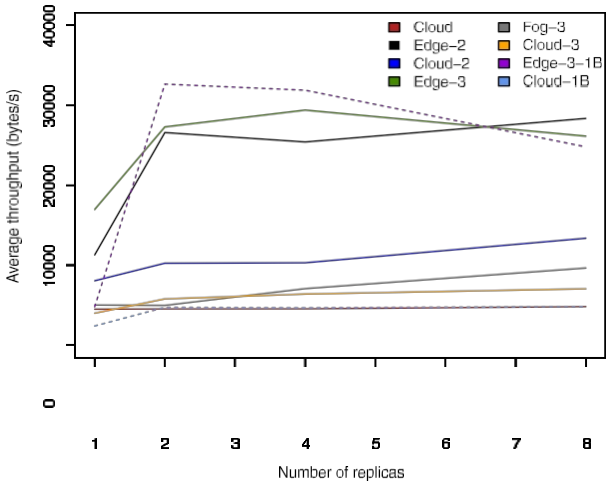


그림 23. 다양한 레플리카 수와 3개의 카프카 브로커를 사용한 평균 추론 처리량.

이 두 번째 테스트의 결과는 Kafka-ML의 새로운 분산 모델 기능으로 인해 시간이 크게 개선되었음을 보여줍니다. 또한 많은 수의 요청을 수용하고 최단 시간 내에 응답을 제공할 수 있는 준비가 되어 있습니다. 세 개의 카프카 브로커를 사용해 고성능 시나리오에서 카프카-ML이 적절하게 작동하는 것을 확인했습니다. 또한 클라우드-사물 간 연속체에서 DDN을 사용하는 것이 클라우드 플랫폼에만 의존하는 것보다 훨씬 더 효율적이라는 것이 입증되었습니다.

C. 테스트 3. 내결함성 및 고가용성 평가

이 마지막 테스트에서는 장애 상황이 발생했을 때 Kafka-ML의 성능이 어떻게 영향을 받는지 평가했습니다. 특히, 배포된 카프카-ML 추론 복제본과 카프카 브로커에 장애가 발생했을 때 내결함성 및 고가용성 특성을 평가했습니다. 테스트에는 8개의 복제본, 전체 클라우드-사물 연속체(엣지-포그-클라우드)를 포괄하는 시나리오 3, 최대 데이터 수집(32개 클라이언트)을 고려하여 3개의 브로커가 있는 기존의 고성능 Kafka-ML 배포가 채택되었습니다. 테스트를 실행하는 동안 추론 구성 요소와 Kafka 브로커를 수동으로 중지하여 이러한 구성 요소의 장애를 시뮬레이션했습니다.

그림 24, 25, 26은 각각 엣지, 포그, 클라우드에서 서로 다른 양의 레플리카가 작동을 멈췄을 때의 추론 응답 시간을 보여줍니다. 결과에 따르면 중단된 레플리카의 수가 증가함에 따라 다음과 같은 응답 시간이 증가합니다.

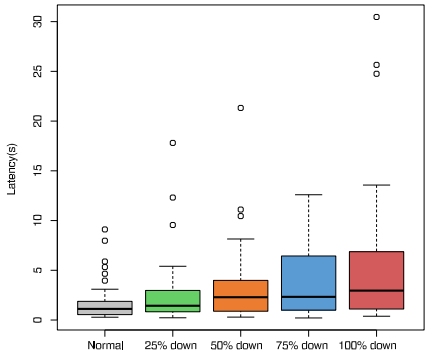


그림 24. 32개의 클라이언트와 서로 다른 수의 레플리카가 다운된 엣지에서의 추론 지연 시간 응답.

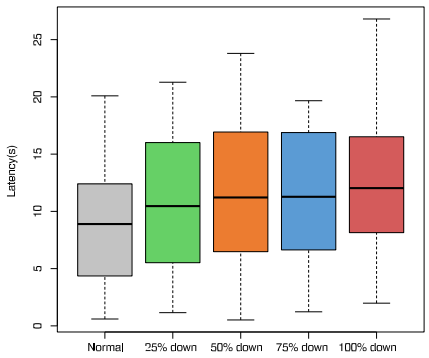


그림 25. 32개의 클라이언트와 서로 다른 수의 레플리카가 다운된 상태에서 Fog의 추론 지연 시간 응답.

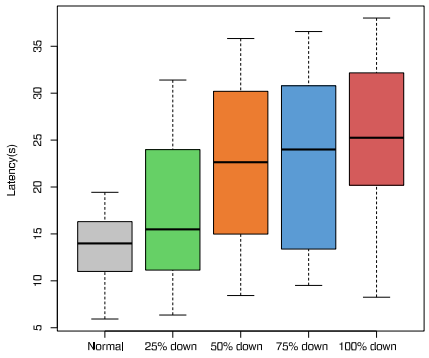


그림 26. 32개의 클라이언트와 서로 다른 수의 복제본이 다운된 클라우드에서의 추론 지연 시간 응답.

카프카-ML 추론도 연구한 세 가지 레이어에서 증가했습니다. 이는 지연 시간도 더 높은 포그 및 클라우드와 같은 최상위 레이어에 더 큰 영향을 미칩니다. 그러나 이 결과는 Kafka-ML이 배포된 추론 모듈의 장애를 최대 100%까지 처리할 수 있음을 보여줍니다. 이는 추론 구성 요소나 서버에 장애가 발생하면 이를 감지하고 매우 짧은 시간 내에 사용

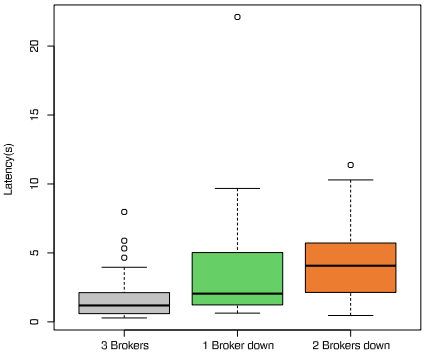


그림 27. 32개의 클라이언트와 서로 다른 아파치 카프카 브로커를 사용한 엣지에서의 추론 지연 시간 응답.

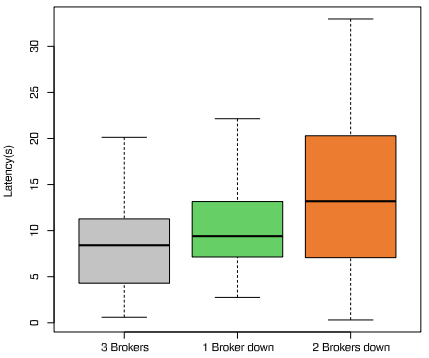
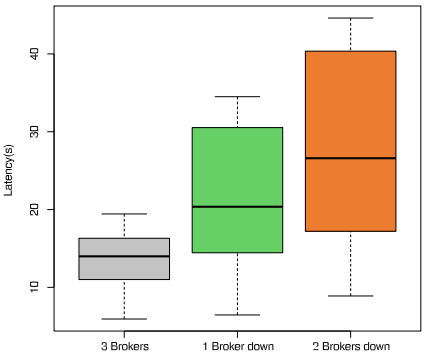


그림 28. 32개의 클라이언트와 서로 다른 아파치 카프카 브로커가 다운된 포그에서의 추론 지연 시간 응답.



가능한 노드에서 추론 구성 요소를 다시 가져오는 Kubernetes에서 제공하는 Kafka-ML 구성 요소의 지속적인 모니터링 덕분에 달성할 수 있습니다. 반면에 그림 27, 28, 29는 각각 엣지, 포그, 클라우드의 응답 시간을 보여줍니다.

그림 29. 클라우드에서 32개의 클라이언트와 서로 다른 아파치 카프카 브로커를 사용한 추론 지연 시간 응답.

를 반환합니다. 추론 모듈과 마찬가지로, Kafka 브로커의 장애도 Kafka-ML에서 처리됩니다. 마찬가지로 모든 경우, 특히 상위 레이어에서 응답 시간이 증가합니다. 그러나 이러한 장애는 지속적인 모니터링을 통해 감지되며 단시간 내에 정상 작동으로 복귀됩니다.

이 테스트 결과는 Kafka-ML이 실행 중에 내결함성과 고가용성 기능을 어떻게 제공하는지 보여줍니다. 특히 계층 간 통신을 담당하는 추론 모듈과 카프카 브로커의 장애를 카프카-ML이 어떻게 성공적으로 처리하는지 확인했습니다,

지연 시간이 약간 증가했습니다. 따라서 이는 연속체를 따라 분산된 모델을 사용하여 지연 시간이 짧고 고가용성 및 내결함성이 필요한 애플리케이션에 대한 Kafka-ML의 실행 가능성을 입증합니다.

VIII. 결론

클라우드-사물 연속체에서 분산 및 심층 신경망의 할당과 관리는 시간에 민감한 요구사항이 있는 애플리케이션에서 큰 관심을 끌고 있습니다. 또한 데이터 스트림 기술과의 조화는 IoT와 같이 현재 혁신적으로 변화하는 데이터 스트림 소스와의 통합을 용이하게 해줍니다. 이러한 이유로 유니티는 데이터 스트림과 ML/AI 프레임워크를 결합하는 오픈 소스 프레임워크인 Kafka-ML을 확 장 하 여 클라우드-사물 간 연속체에서 분산된 DNN 파이프라인의 배포 및 관리를 지원합니다. 여기에는 분산형 ML 모델의 설계, 데이터 스트림을 통한 학습, 내결함성과 고가용성을 제공하는 최 첨 단 컨테이너화 기술과 데이터 스트림 수집을 통해 컨티뉴엄에 배포하는 등 ML/AI 애플리케이션의 파이프라인이 포함됩니다. 또한 Kafka-ML 덕분에 오픈 소스 솔루션에서 모델 공유, 메트릭 평가, 모델 다운로드 및 데이터 스트림 관리도 가능합니다. 정확도가 허용 가능한 수준일 때 최대한 빠른 응답을 제공하기 위해 BranchyNet을 기반으로 한 조기 종료가 채택되었습니다. 결과적으로, 엣지 클라우드 및 엣지 포그 클라우드 배포에서 이 프레임워크의 검증은 클라우드 배포에 대해서만 제안된 접근 방식이 개선되었음을 보여줍니다.

동적 DDNN 파티셔닝은 클라우드-사물 연속체에서 적시에 적재적소에 DDNN 계층을 할당하여 배포 시나리오(네트워킹 + 하드웨어)의 현재 상태에 맞게 DDNN을 조정하기 위한 향후 작업으로 검토될 예정입니다. 이 작업은 Kafka-ML에서 향후 계획된 작업 중 하나이기도 합니다. 따라서 분산형 트레이닝을 지원하고 트레이닝 단계를 가속화하며, 텐서플로우를 넘어 더 많은 ML/AL 프레임워크를 지원하고, 중요한 의사결정을 위한 온디바이스 추론 관리 및 배포를 지원함으로써 Kafka-ML 프레임워크를 지속적으로 개선해 나갈 것입니다.

참고 자료

[1] S. Sagiroglu와 D. Sinanc, "빅 데이터: A review," in *Proc. Conf. Collaboration Technol. Syst. (CTS)*, San Diego, CA, USA, May 2013, pp. 42-47.

[2] Y. Lu, "인공 지능: 진화, 모델, 애플리케이션 및 미래 동향에 대한 조사," *J. Manage. Anal.*, vol. 6, no. 1, pp. 1-29, Jan.

[3] G. Stringhini, C. Kruegel, 및 G. Vigna, "소셜 네트워크에서 스팸 발송자 탐지", *26th Annu. Comput. Secur. Appl. Conf.*, 뉴욕, NY, USA, 2010, 1-9쪽, 도이: 10.1145/1920261.1920263.

[4] E. Mariconti, G. Suarez-Tangil, J. Blackburn, E. De Cristofaro, N. Kourtellis, I. Leontiadis, J. L. Serrano 및 G. Stringhini, "'You know what to do" 조직적인 증오 공격의 표적이 된 YouTube 비디오의 사전 탐지," *Proc. ACM Hum.-Comput. Interact.*, 3권, 11월, 2019, pp. 1-15, 도이: 10.1145/3359309.

[5] A. Abdelaziz, M. Elhoseny, A. S. Salama 및 A. M. Riad, "클라우드 컴퓨팅 환경에서 의료 서비스 개선을 위한 기계 학습 모델", *Meas. J. Int. Meas.*, vol, pp. 117-128, Apr. [온라인]. 이용 가능: https://www.sciencedirect.com/science/article/pii/S0263224118300228

[6] M. Chen, Y. Hao, K. 황, L. Wang, "의료 커뮤니티의 빅데이터를 통한 머신러닝을 통한 질병 예측", *IEEE Access,* 5권, 8869-8879면, 2017.

[7] L. Piyathilaka, D. Preethichandra, U. Izhar 및 G. Kahandawa, "심층 전송 학습을 사용한 실시간 콘크리트 균열 감지 및 인스턴스 분할", *Eng. Proc.*, vol. 2, no. 1, p. 91, 2020. [온라인]. Available: https://www.mdpi.com/2673-4591/2/1/91

[8] Y. Bao, Z. Tang, H. Li, Y. Zhang, "구조적 상태 모니터링을 위한 컴퓨터 비전 및 딥러닝 기반 데이터 이상 탐지 방법", *Struct. 헬스 모니터,* 18권 2호, 401-421쪽, 2019.

[9] L. Hui와 T. Chin-Chung, "정밀 교육을 위한 기계 학습 접근법 사용에 대한 검토", *J. Educ. Technol. Soc.*, vol. 24, no. 1,
pp. 250-266, 2021.

[10] M. 디아즈, C. 마르틴, B. 루비오, "사물 인터넷과 클라우드 컴퓨팅 통합의 최첨단, 도전 과제 및 미해결 문제", '사물 인터넷과 클라우드 컴퓨팅의 통합,"
J. Netw. Comput. 응용, 67권, 99-117쪽, 2016년 5월.

[11] *사물 인터넷 개요.* Accessed: Feb. 20, 2021. [온라인]. 사용 가능: https://emarsonindia.com/wp-content/uploads/2020/02/Internet-of-Things.pdf

[12] G. 응우옌, S. 드루골린스키, M. 보박, V. 트란, A. 엘 가르시아, I. 에레디아,
P. Malík, 및 L. Hluchý, "대규모 데이터 마이닝을 위한 머신러닝 및 딥러닝 프 레 임 워 크 와 라이브러리: 설문 조사," *Artif. Intell. Rev.*, 52, 1권, 77-124쪽, 2019.

[13] M. 아바디. (2015). *텐서 플로우 : 이종 유전자 시스템에 대한 대규모 기계 학습.* [온라인]. Available: https://www.tensorflow.org/

[14] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan,
T. Killeen, 및 Z. Lin, "Pytorch: 명령형 스타일의 고성능 딥러닝 라이브러리", *Proc. Adv. Neural Inf. Process. Syst.*, 2019,
pp. 8024-8035. [온라인]. 사용 가능: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[15] C. Martín, P. Langendoerfer, P. S. Zarrin, M. Díaz, 및 B. Rubio, "Kafka-ML: 데이터 스트림을 ML/AI 프레임워크와 연결하기," *Future Gener. Comput. Syst.*, vol. 126, pp. 15-33, Jan. [온라인]. 제공: https://www.sciencedirect.com/science/article/pii/S0167739X21002995

[16] *아파치 카프카.* Accessed: May 13, 2020. [온라인]. Available: http://kafka. apache.org/

[17] C. M. Fernandez, M. D. Rodriguez, 및 B. R. Munoz, "사물인터넷의 엣지 컴퓨팅 아키텍처", *Proc. Symp. Real- Time Distrib. Comput. (ISORC)*, 싱가포르, May 2018, 99-102쪽.

[18] D. R. Torres, C. Martín, B. Rubio 및 M. Díaz, "클라우드-사물 연속체를 통한 DDNN 추론을 위한 Kafka-ML 기반 오픈 소스 프레임워크", *J. Syst. Archit.*, vol. 118, Sep. 2021, Art.

[19] S. Teerapittayanon, B. McDanel 및 H. T. Kung, "클라우드, 에지 및 최종 장치를 통한 분산 심층 신경망", *Proc. Conf. Distrib. Comput. Syst. (ICDCS)*, 미국 조지아주 애틀랜타, June, 2017,
pp. 328-339.

[20] *쿠버네티스.* Accessed: Jan. 9, 2021. [온라인]. Available: https://www.kubeflow.org/

[21] S. Tuli, N. Basumatary, 및 R. Buyya, "EdgeLens: 통합 IoT, 포그 및 클라우드 컴퓨팅 환경에서의 딥러닝 기반 객체 감지", *Proc. Conf. Inf. Syst. Comput. (ISCON)*, 인도 마투라, 2019년 11월, 496-502쪽.

[22] S. 툴리, N. 바수마타리, S. S. 길, M. 카하니, R. C. 아리아, G. S. 원더, 및 R. Buyya, "HealthFog: 통합 IoT 및 포그 컴퓨팅 환경에서 심장 질환 자동 진단을 위한 앙상블 딥러닝 기반 스마트 헬스케어 시스템", *Future Gener. Comput. Syst.*, vol,
pp. 187-200, Mar.

[23] A. Javed, J. Robert, K. Heljanko, 및 K. Främling, "IoTEF: 내결함성 IoT 애플리케이션을 위한 연합 에지 클라우드 아키텍처," *J. Grid Com- put.*, 10권, 1-24쪽, 2020년 1월.

[24] M. D. da Assunção, A. da S. Veith, 및 R. Buyya, "분산 데이터 스트림 처리 및 엣지 컴퓨팅: 자원 탄력성 및 향후 방향에 대한 조사," *J. Netw. Comput. Appl.*, vol. 103, pp. 1-17, Feb.

[25] N. Kourtellis, G. D. F. Morales, 및 A. Bifet, "아파치 사모아를 이용한 대규모 데이터 스트림 학습", *진화하는 환경에서 데이터 스트림으로부터 학습, 진화하는 환경에서 데이터 스트림으로부터 학습.* Cham, Switzerland: Springer, 2019, 177-207쪽.

[26] S. Teerapittayanon, B. McDanel, 및 H. T. Kung, "BranchyNet: 심층 신경망에서 조기 종료를 통한 빠른 추론", *Proc. Conf. Pattern Recognit. (ICPR)*, 멕시코 칸쿤, 2016년 12월, 2464-2469쪽.

[27] L. A. Steffenel, M. Kirsch Pinheiro, 및 C. Souveyet, "에지 컴퓨팅에서 불균형한 리소스 및 통신의 영향 평가", *Pervas. 모바일 컴퓨팅,* 71 년 2 월, 2021 년 2 월, 예술 번호 101321. [온라인]. Available: http://www.sciencedirect. com/science/article/pii/S1574119220301450

[28] T. Mohammed, C. Joe-Wong, R. Babbar 및 M. D. Francesco, "적응형 DNN 파티셔닝 및 오프로딩을 통한 분산 추론 가속", *Proc. IEEE INFOCOM - IEEE Conf. Comput. Commun.*, 베이징, 중국, 7월, 2020, 854-863쪽.

[29] R. G. Pacheco와 R. S. Couto, "BranchyNet 파티셔닝을 이용한 추론 시간 최적화", 2020, *arXiv:2005.04099*. [온라인]. Available: http://arxiv.org/abs/2005.04099

[30] J. Zhou, Y. Wang, K. Ota, M. Dong, "AAIoT: IoT 시스템에서 인공 지능 가속화," *IEEE Wireless Commun. Lett.*, vol. 3, pp. 825-828, Jun.

[31] E. D. 코닌크, S. 보헤즈, S. 르루, T. 베르벨렌, B. 반케르스빌크, P. Simoens, and B. Dhoedt, "DIANNE: 이기종 분산 인프라에서 심층 신경망을 설계, 훈련 및 배포하기 위한 모듈식 프레임워크", *J. Syst. Softw.*, vol. 141, pp. 52-65, Jul.

[32] C. Hu, W. Bao, D. Wang, 및 F. Liu, "에지에서의 추론 가속을 위한 동적 적응형 DNN 수술", *Proc. Comput. Commun.*, 프랑스 파리, 2019년 4월, 1423-1431쪽.

[33] R. G. Pacheco와 R. S. Couto, "BranchyNet 파티셔닝을 이용한 추론 시간 최적화,", *IEEE Symp. Comput. Commun. (ISCC)*, 프랑스 렌, 7월, 2020, 1-6쪽.

[34] E. Li, L. Zeng, Z. Zhou 및 X. Chen, "Edge AI: 엣지 컴퓨팅을 통한 온디맨드 가속 심층 신경망 추론", *IEEE Trans. 무선 통신,* 19권 1 호, 447-457쪽, 2020 년 1 월.

[35] J. Soldani, D. A. 탐부리, W.-J. 반 덴 호벨, "마이크로서비스의 득과 실 '' : 체계적인 회색 문헌 검토," *J. Syst. Softw.*, 146권, 215-232쪽, 2018년 12월.

[36] A. Krizhevsky, I. Sutskever, 및 G. E. Hinton, "심층 컨볼루션 신경망을 이용한 ImageNet 분류", *Commun. ACM*, vol, pp. 84-90, 2017년 5월, 도이치: 10.1145/3065386.

[37] A. 크리제프스키, "작은 이미지에서 여러 층의 특징 학습하기", 참조. 석사 학위 논문, 토론토 대학교, 토론토, 온타리오, 캐나다, 2009.

알레한드로 카르네로는 2020년 스페인 말라가 대학교에서 소프트웨어 공학 학사 학위를 받았으며, 현재 소프트웨어 공학 및 인공지능 석사 과정을 밟고 있습니다. 2020년부터는 말라가 대학교 ERTIS 연구 그룹에서 연구 조교 로 일하고 있습니다. 그의 연구 관심 분야는 분산형 딥 머신 러닝 과 IoT 분야입니다.

다니엘 R. 토레스는 2017년과 2018년에 각각 컴퓨터 공학 디플로마 및 학사 학위를 받았습니다. 현재 말라가 대학교 ERTIS 연구소에서 컴퓨터 공학 석사 과정을 밟고 있습니다. 다양한 기술 회사에서 소프트웨어 엔지니어로 근무하며 스크립팅, 웹 , iOS와 같은 소프트웨어 앱 개발 및 구현, 데이터 분석 업무를 주로 담 당 했 습 니 다 . 그는 현재 말라가 대학교 ERTIS 연구실. 주요 연구 분 야 는 인공 지능 응용, 물리학, 신경과학입니다. 컴퓨터 및 네트워크 시스템 관리 분야에서 고등 교육 및 직업 훈련 자격증을 취득했습니다.

다니엘 가리도는 1999년과 2006년에 각각 스페인 말라가 대학교에서 컴퓨터 공학 석사와 박사 학위를 받았습니다. 2006년부터는 특히 소프트웨어 엔지니어링 분야에서 분산 프로그래밍, 시뮬레이션, 실시간 시스템과 관련된 다양한 공공 및 민간 프로젝트에 참여했습니다. 2007년부터 2009년까지 말라가 대학교 언어 및 컴퓨터 과학과의 조교수로 재직하며 다음과 같은 업무를 수행했습니다.
2009년부터 부교수로 재직 중입니다. 그는 말라가에 있는 중요 시스템용 소프트웨어 회사인 Softcrits의 창 립 자 중 한 명 입 니 다 . 그는 국제적으로 권위 있는 학술지와 이 분야의 저명한 학술대회에 여러 편의 논문을 발표했습니다.

마누엘 디아즈는 현재 말라가 대학교 컴퓨터 과학과의 정교수이며, ERTIS 연구 그룹 책임자이자 ITIS 소프트웨어 연구소의 회원이기도 합니다. 지난 몇 년 동안 그의 주요 연 구 분야는 WSN 및 모니터링 시스템, 특히 에너지 모니터링(FP7 e-balance 프로젝트), 수자원 인프라 모 니 터 링 (FP7 SAID 프로젝트), 에너지 효율적 건물(FP7 SEEDS)에 중점을 두었습니다. 그는 다음 분야에서 협력했습니다.
테크나톰, 텔레포니카, 인드라, 아벤고아 등 다양한 기업과의 기술 이전 프로젝트를 다수 수행했습니다. 그는 FP6 SMEPP 프로젝트의 코디네이터이자 WSAN4CIP(ICT FP7)에서 UMA의 수석 연구원이었습니다. 또한 스핀오프 소프트크릿의 공동 설립자이자 연구 개발 부서의 책임자입니다. 그의

크리스티안 마르틴은 스페인 말라가 대학교에서 2014년, 2015년, 2018년에 각각 컴퓨터 공학 석사, 소프트웨어 공학 및 인공지능 석사, 컴퓨터 과학 박사 학위를 받았습니다. 그는 다양한 기술 기업에서 소프트웨어 엔지니어로 근무하며 RFID 기술 및 소프트웨어 개발을 담당했습니다. 현재 말라가 대학교에서 박사후 연구원으로 재직 중입니다. 또한 ITIS Software의 회원이기도 합니다.

말라가 대학교 연구소. 그의 연구 분야는 사물 인터넷과 클라우드/포그/에지 컴퓨팅의 통합, 머신 러닝, 구조적 상태 모니터링, IoT 신뢰성 등입니다.

바르톨로메 루비오는 말라가 대학교에서 1990년과 1998년에 각각 컴퓨터 공학 석사 및 박사 학위를 받았습니다. 1991년부터 2000년까지 말라가 대학교 언어 및 컴퓨터 과학과의 조교수로 재직했으며, 2001년부터는 부교수로 재직하고 있습니다. 그는 분산 및 병렬 프로그래밍과 조정 모듈 및 언어 분야에서 일해 왔습니다. 그는 특히 무선 센서 및 액터 네트워크, 사물 인터넷과 클라우드 컴퓨팅의 통합 분야를 연구하고 있습니다. 그는 말라가대학교 소프트웨어 엔지니어링 그룹이 설립될 때부터 멤버로 활동해 왔습니다. 말라가대학교 ITIS 소프트웨어 연구소의 회원입니다.