chapter 1

엣지 컴퓨팅을 위한 스케줄링 기술 동향

장수민 ‖ 한국전자통신연구원 책임연구원
김대원 ‖ 한국전자통신연구원 책임연구원
최현화 ‖ 한국전자통신연구원 책임연구원
차재근 ‖ 한국전자통신연구원 책임연구원
김선욱 ‖ 한국전자통신연구원 책임연구원/PL

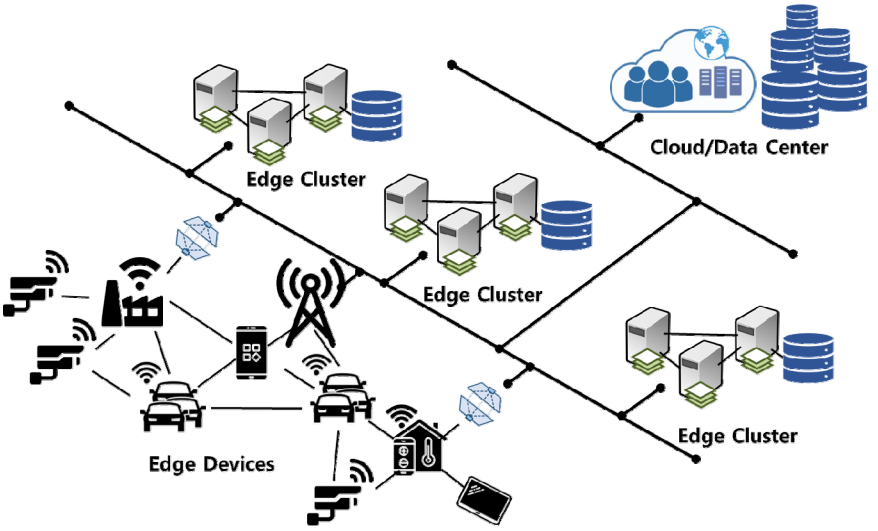
최근에 사물인터넷, AR/VR, 자율주행자동차 관련 서비스들이 대규모 데이터에 대한 빠른 연산 처리를 위해 엣지 컴퓨팅 기술을 사용하고 있다. 이러한 엣지 컴퓨팅 기술에서는 한정된 자원을 효과적으로 분산 및 배치/할당하는 스케줄러의 역할이 매우 중요하다. 본 고에서는 엣지 컴퓨팅에 사용되는 주요 스케줄링 기술들에 대한 특징을 살펴보고자 한다. 이와 함께 초저지연 서비스와 지능형 서비스를 동시에 지향하는 GEdge 플랫폼의 스케줄러를 살펴봄으로써 효율적인 엣지 간 협업과 최적 자원 배치를 위한 스케줄링 기술에 대한 이해를 돕고자 한다.

I. 서론

1. 엣지 컴퓨팅 소개

엣지 컴퓨팅 서비스는 산업현장이나 사무 환경에서 수집된 대규모 데이터가 클라우드에

* 본 내용은 장수민 책임연구원(☎ 042-860-0724, jsm@etri.re.kr)에게 문의하시기 바랍니다.
** 본 내용은 필자의 주관적인 의견이며 IITP의 공식적인 입장이 아님을 밝힙니다.
***이 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2020-0-00116, 10msec 미만의 서비스 응답 속도를 보장하는 초저지연 지능형 클라우드 엣지 SW 플랫폼 핵심 기술 개발)"



〈자료〉 한국전자통신연구원 자체 작성

[그림 1] 엣지 컴퓨팅 서비스 실행 환경

있는 원격 서버로 가기 전에 엣지 플랫폼에서 대규모로 분석하여 처리할 수 있는 환경을 제공한다.

최근에 삼성전자와 IBM이 엣지 컴퓨팅과 프라이빗 5G 네트워크 결합과 관련하여 제휴를 맺어 기업들의 제조 현장 자동화를 지원하기 위해 엣지 컴퓨팅 서비스를 제공한다는 뉴스 기사 발표에서도 알 수 있듯이, 엣지 컴퓨팅은 우리 삶 현장에서 쉽게 접할 수 있는 키워드가 되었다[1].

이러한 엣지 컴퓨팅 서비스 실행 환경은 [그림 1]과 같이 크게 3가지로 구성된다.

- 클라우드: 기존 Public/Private 클라우드나 데이터센터
- 엣지 클러스터: 지역별로 분산된 엣지 간의 협업을 통해 서비스가 가능
- 엣지 디바이스: 공장의 자동화 단말기/장비, IOT 장비, 홈서비스 단말기, 핸드폰

특히, 엣지 컴퓨팅 기술은 사용자의 몰입감을 최적화하기 위해 초저지연 연산 처리 및 높은 안정성 및 큰 네트워크 대역폭을 요구하는 가상/증강현실(VR/AR)에 매우 효과적이다[2]. 또한 자율주행자동차의 경우에는 빠른 응답 시간이 매우 중요하기 때문에 장치 자체에는 클라우드 연결에 의존하지 않고 이미지를 분석하고 분류하는 AI 서비스에 적합한 엣지 컴퓨팅 환경을 요구하고 있다[3]-[5]. 이러한 AI 서비스에 적합한 대표적인 엣지 컴

퓨팅 플랫폼으로 NVIDIA의 'EGX'[6]라는 실시간 AI 엣지 컴퓨팅 플랫폼이 있다.

2. 엣지 컴퓨팅의 스케줄러

기본적인 엣지 컴퓨팅의 스케줄러의 역할은 요청된 서비스에 해당하는 컨테이너를 어느 호스트에 할당할지에 대한 물음의 답을 제공하는 것이다. 즉, 스케줄러의 역할은 기본적으로 연산 노드들이 여러 개 있을 때 서비스를 할당하기에 가장 알맞은 연산 노드를 찾는 것이다. 엣지 컴퓨팅은 기존 중앙 집중형 클라우드 기반의 서비스를 제공하는 솔루션을 엣지 플랫폼에서 활용할 수 있도록 엣지와 엣지 간에 협업하고 연동하는 인터페이스를 제공하여 엣지 기반 솔루션의 신속한 전환을 가능하게 하여 새로운 서비스를 제공하는 특징을 가지고 있다. 그래서, 이러한 엣지 컴퓨팅을 위한 스케줄러는 사용자로부터 요청된 서비스들을 하나의 엣지 클러스터에 국한되지 않고 클라우드 및 지리적으로 분산된 엣지 클러스터들을 통합하여 제한된 리소스에 할당 및 배치하는 중요한 역할을 담당해야 한다.

엣지 컴퓨팅을 위한 스케줄러의 주된 역할은 앞에서 언급한 기본적인 스케줄러의 기능뿐만 아니라 다양한 도메인 서비스에 특화된 클라우드-엣지-단말 간 협업형 엣지 서비스를 효과적으로 초저지연으로 처리하고 이동형 서비스의 협업이 가능하도록 제한된 자원을 효과적으로 분산 및 할당하는 것이다.

본 고에서는 엣지 컴퓨팅을 위한 스케줄링 핵심 기술인 컨테이너 오케스트레이션에 관련된 연구 내용과 엣지 컴퓨팅에 관련된 플랫폼의 스케줄러 개발에 관련된 연구도 언급하고자 한다. 이를 위해, 먼저 II장에서는 컨테이너 오케스트레이션의 스케줄러들의 특징을 먼저 살펴보고, III장에서는 엣지 컴퓨팅에 관련된 플랫폼인 GEdge 플랫폼의 글로벌 스케줄러 핵심 기술을 설명하고, IV장에서 본 고의 결론을 제시한다.

II. 컨테이너 오케스트레이션의 스케줄러

엣지 컴퓨팅은 애플리케이션 또는 서비스 시스템 관리를 자동화하고 기존 환경에 비해 시스템 안정성을 높이고 운영/관리 업무의 효율화와 비즈니스 요구에 대한 빠른 응대, 시스템 확장이 용이한 컨테이너 기반의 가상화 기술을 적극적으로 활용하고 있다. 이러한

컨테이너 기반의 가상화 기술은 그 적용되는 서비스 영역을 확대하여 다양한 응용 서비스를 컨테이너화하고 그렇게 정의된 컨테이너를 쉽고 빠르게 배포, 확장, 관리를 자동화해주는 플랫폼 기술로 발전되었다. 이러한 플랫폼 기술을 구체적으로 컨테이너 오케스트레이션이라 명칭하여 사용하고 있으며 최근에 널리 사용되고 있는 대표적인 컨테이너 오케스트레이션 툴로는 Kubernetes, Docker Swarm, Apache Mesos가 있다.

이러한 컨테이너 오케스트레이션 플랫폼에서 스케줄러는 클러스터의 자원들을 효율적으로 사용하고 사용자에게 제공되는 배치의 제약 혹은 제한 기능이 작동되어야 하고 스케줄링 과정에서는 공정성을 가지고 요청한 작업을 신속하게 배치 및 할당하는 중요한 역할을 담당하고 있다. 본 II장에서는 컨테이너 오케스트레이션을 위한 주요 스케줄링인 모놀리식 스케줄링(Monolithic Scheduling), 2단계 스케줄링(Two-level Scheduling), 공유 상태 스케줄링(Shared-state Scheduling)에 대한 개념을 소개하고 이에 관련된 Docker Swarm, Kubernetes 스케줄러의 상세 개념들을 정리하였다.

1. 컨테이너 오케스트레이션의 주요 스케줄링

가. 모놀리식 스케줄링

모놀리식 스케줄링(Monolithic Scheduling)은 모든 요청을 처리하는 단일 스케줄링 에이전트로 구성되며 일반적으로 고성능 컴퓨팅에 사용된다. 모놀리식 스케줄링은 일반적으로 모든 수신 작업에 대해 단일 알고리즘 구현을 적용하므로 작업 유형에 따라 서로 다른 스케줄링 논리를 실행하기가 어렵다. 많은 스케줄링 기능을 애플리케이션 구성 요소에 위임하는 Apache Hadoop YARN이나 Docker Swarm 등이 애플리케이션 마스터의 리소스 요청을 단일 글로벌 스케줄러로 전송하는 모놀리식 스케줄러 방식을 사용하고 있다.

나. 2단계 스케줄링

운영체계에서 2단계 스케줄링(Two-level Scheduling)은 메모리에서 실행할 프로세스 중에서 선택할 수 있는 하위 레벨 스케줄러(Lower-level Scheduler)와 메모리에서 프로세스를 스왑 인/아웃을 하는 상위 레벨 스케줄러(Higher-level Scheduler)로 구성된 스케줄러이다. 이와 비슷하게 컨테이너 오케스트레이션에서 2단계 스케줄링은 중앙 코디네

이터를 사용하여 각 하위 클러스터가 보유할 수 있는 리소스 수를 결정하고, 각 서브 스케줄러에 대한 리소스 할당을 동적으로 조정하는 방식이다. 중앙 코디네이터가 각 서브 스케줄러에 제공하는 리소스의 순서와 크기를 선택하고 한 번에 하나의 프레임워크에만 주어진 리소스를 제공하여 각 서브 스케줄러 간에 충돌을 피하면서 리소스 사용의 공정성을 추구하는 방식이다. 그러나 한 번에 하나의 프레임워크만 리소스를 검사하므로 동시성 제어 측면에서는 단점을 가지고 있다.

다. 공유 상태 스케줄링

공유 상태 스케줄링(Shared-state Scheduling)은 각 스케줄러에 전체 클러스터에 대한 전체 액세스 권한을 부여하여 모든 방식으로 경쟁할 수 있도록 하는 방식이다. 모든 리소스 할당 결정이 스케줄러에서 이루어지므로 중앙 리소스 할당자가 없다. 이러한 방식은 각 스케줄러를 구현하는 측면에서 독립적이고 스케줄러의 전체 할당 상태를 공유함으로써 많은 스케줄러로 확장할 수 있으며 자체 스케줄링 정책 사용이 가능하다. 그래서 이 방식은 동시성 제어 측면에서는 매우 효과적이다.

주요 컨테이너 오케스트레이션의 스케줄링 방식을 구분해 보면 [표 1]과 같이 모놀리식 스케줄링은 Docker Swarm에서 사용하고 있고, 2단계 스케줄링은 컴퓨터 클러스터를 관리하기 위한 오픈 소스 프로젝트인 Mesos가 채택하고 있다. 최근에 컨테이너 오케스트레이션 툴로 가장 많이 사용되고 있는 Kubernetes와 Docker Swarm mode, Nomad가 사용하는 스케줄링 방식이 공유 상태 스케줄링이다.

[표 1] 주요 컨테이너 오케스트레이션의 스케줄러 구분

스케줄링	컨테이너 오케스트레이션
Monolithic	Docker Swarm
Two-level	Mesos
Shared-state	Docker Swarm mode, Kubernetes, Nomad

〈자료〉 Armand Grillet, “Comparison of Container Schedulers,” 2016.

2. Docker Swarm 스케줄러

Docker Swarm은 Docker 애플리케이션을 실행하고 클러스터에서 함께 결합하도록

구성된 물리적 또는 가상 머신의 그룹이다. Docker Swarm은 클러스터 상태를 관리하는 매니저 노드(Manager Node)와 매니저 노드의 명령을 받아 컨테이너를 생성하고 상태를 점검하는 워커노드(Worker Node)로 구성되어 있다.

가. Docker Swarm 스케줄러 필터

Docker Swarm 스케줄러 필터는 Docker 및 컨테이너의 특성에 맞게 노드를 선택하기 위한 필터이다. 필터에는 두 가지 종류가 있는데 Node Filter는 각 노드의 특성에 맞게 자원을 할당하고, Container Filter는 컨테이너의 특성에 맞게 자원을 할당하기 위해서 사용된다.

(1) Node Filters

- Constraint: Docker Daemon을 실행시킬 때, Label 옵션을 주고 Docker Daemon의 Label에 따라서 컨테이너를 할당한다.
- Healthy: Swarm 클러스터의 노드가 Healthy 상태에 있는지 확인하여 해당 노드에 컨테이너를 할당한다.
- Containerslots: 노드에 할당할 수 있는 컨테이너의 수를 제한하는 필터이다. n개로 설정한 노드에 n개의 컨테이너가 할당되었다면 더 이상 할당하지 않는다.

(2) Container Filters

- Affinity: 특정 컨테이너나 이미지, 특정 라벨이 붙은 컨테이너가 존재하는 노드를 고르는 것이다.
- Dependency: 특정 컨테이너/Link/볼륨을 공유하거나, 네트워크 옵션에서 특정 Network Stack을 사용해야 할 때 사용한다. 즉, 연관이 있는 컨테이너들을 같은 노드에 위치시킴으로써 정상적으로 작동한다.
- Port: 특정 포트가 사용 가능한 노드를 찾는 것이다. 이는 호스트의 포트를 의미하며, Swarm은 해당 포트가 다른 컨테이너 혹은 프로세스에 의해 점유되지 않은 노드를 선택한다.

나. Docker Swarm 스케줄러 우선순위

- Spread: 모든 Swarm 노드에 골고루 분포하도록 하는 옵션으로 컨테이너의 수가

적은 노드에 컨테이너를 할당하는 방식이다.

- Binpack: 자원을 효율적으로 사용한다는 측면에서 컨테이너를 할당한다. 여러 노드에 컨테이너 자원을 분산시키는 것이 아니라 한 노드씩 자원을 최대한 채워가는 형태로, 새롭게 할당해주어야 할 컨테이너를 위해 준비하는 형태이다.
- Random: 임의의 노드를 선정하여 컨테이너를 생성하는데, 디버깅 용도로 사용된다.

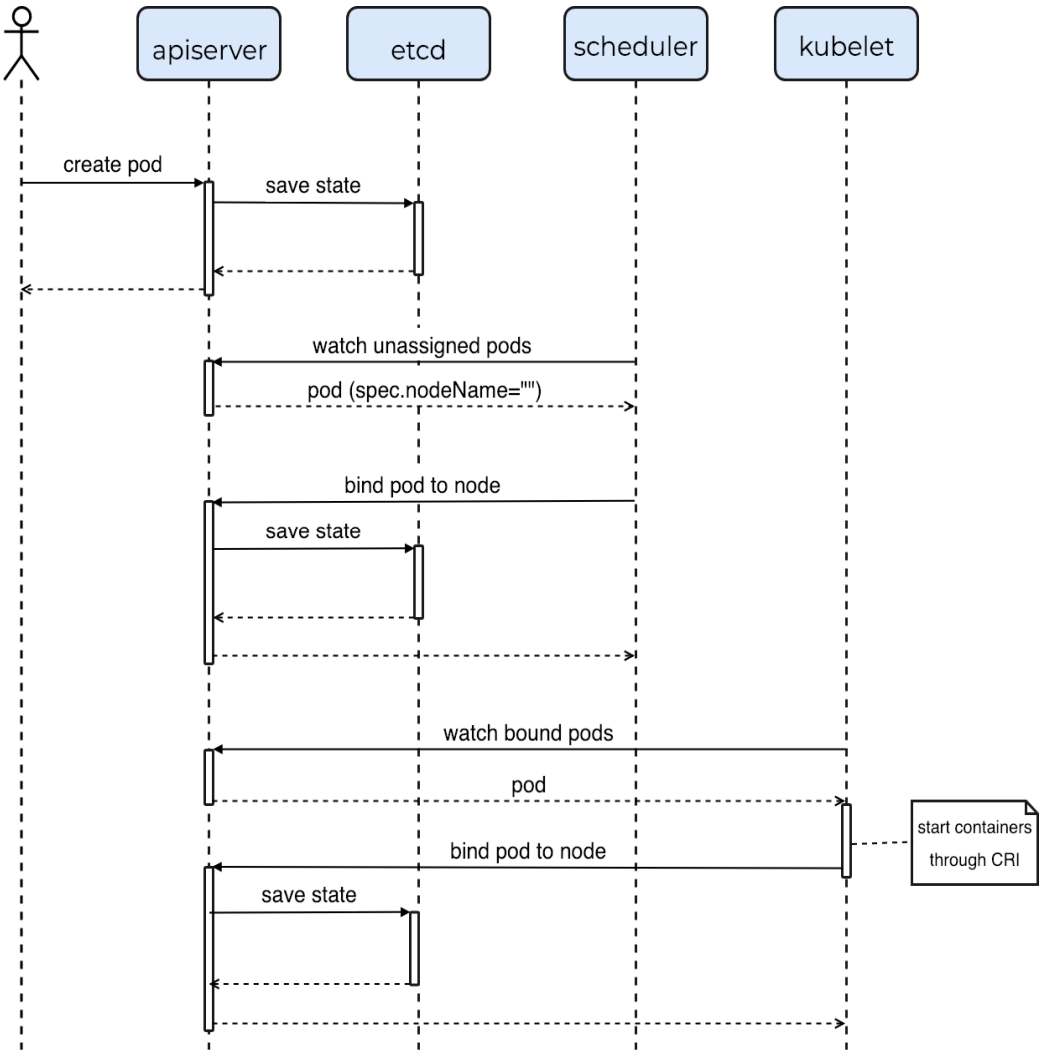
3. Kubernetes 스케줄러

Kubernetes 스케줄러는 모든 리소스 할당 결정이 중앙 리소스 할당자가 없이 스케줄러에서 이루어지는 공유 상태 스케줄링(Shared-State Scheduling) 방식으로 처리된다[7]. 새로운 Pod 생성이 요청되면, 스케줄러는 Pod을 배치할 노드를 선정하여 Pod을 배치하는데 노드 필터링 과정과 노드 순위 지정 과정, 두 가지 과정을 통해 노드를 선정한다.

가. Kubernetes 스케줄러 처리 절차

Kubernetes 스케줄러 동작은 공유 상태 스케줄링으로 [그림 2]와 같은 순서로 새로운 Pod을 할당한다[9]. 특히, 스케줄링으로 발생되는 모든 상태 정보는 etcd에 상태를 기록/공유하면서 처리하는 특징을 갖고 있다.

① kubectl apply 등의 명령어를 통해 사용자의 Pod 생성 요청이 kube-apiserver에 제출되면 kube-apiserver는 etcd에 Pod의 정보를 저장한다.
② Pod이 위치한 노드의 정보인 nodeName의 값이 설정되지 않은 상태로 Pod의 정보를 저장한다.
③ kube-scheduler는 이러한 정보를 watch 과정을 하고 있다가 Pod의 nodeName이 비어 있는 상태라는 것을 감지한다.
④ 해당 Pod을 할당하기 위한 적절한 노드를 찾는, 일종의 스케줄링 작업을 진행한다.
⑤ 적절한 노드를 찾았다면, kube-scheduler는 그 정보를 kube-apiserver에게 전달한다.
⑥ Pod의 직접 생성은 kubelet에서 처리한다.
⑦ Pod의 상태 정보를 etcd에 업데이트한다.



〈자료〉 Banzai Cloud, Writing custom Kubernetes schedulers, 2019.

[그림 2] Kubernetes 스케줄러 처리 흐름도

나. Kubernetes 스케줄러 필터

Kubernetes 스케줄러 필터 옵션은 [표 2]의 요소들을 고려하여 요구 조건에 적합하지 않은 노드들을 모두 배제한 후 남은 노드들을 기초하여 할당하게 된다.

[표 2] Kubernetes 스케줄러 필터

필터(Predicates)	설명
PodFitsHostPorts	노드에 Pod가 요청하는 사용 가능한 포트(네트워크 프로토콜 종류)가 있는지 확인
PodFitsHost	Pod가 호스트 이름으로 특정 노드를 지정하는지 확인
PodFitsResources	노드에 Pod의 요구사항을 충족하는 사용 가능한 리소스(예; CPU 및 메모리)가 있는지 확인
MatchNodeSelector	Pod의 노드 선택기가 노드의 라벨과 일치하는지 확인
NoVolumeZoneConflict	해당 스토리지에 대한 장애 영역 제한을 고려하여 Pod가 요청하는 볼륨을 노드에서 사용할 수 있는지 확인
NoDiskConflict	요청한 볼륨과 이미 마운트된 볼륨으로 인해 Pod가 노드에 맞을 수 있는지 확인
MaxCSIVolumeCount	연결해야 하는 CSI 볼륨 수와 구성된 제한을 초과하는지를 확인
CheckNodeMemoryPressure	노드의 메모리가 부족한지 확인
CheckNodePIDPressure	노드의 프로세스 ID가 부족한지 확인
CheckNodeDiskPressure	노드에 스토리지 압력(가득 찬 또는 거의 가득 찬 파일 시스템)이 있는지 확인
CheckNodeCondition	네트워킹을 사용할 수 없거나 kubelet이 Pod를 실행할 준비가 되는지 등과 같은 조건을 확인
PodToleratesNodeTaints	Pod의 노드 내결함성 오염(taint)을 허용할지 확인
CheckVolumeBinding	요청하는 볼륨의 바인딩이 가능한지 확인

〈자료〉 CNCF(Cloud Native Computing Foundation), “Predicates of Scheduling Policies on Kubernetes,” 2021.

다. Kubernetes 스케줄러 우선순위

Kubernetes 스케줄러 우선순위는 노드 순위 지정 과정을 위한 특정 우선순위 계산 함수들에 의해 우선순위에 대한 점수를 책정하여 점수가 가장 높은 순서대로 노드를 선별한다. 여러 가지의 우선순위가 있는데 [표 3]과 같이 매우 다양한 형태를 지원하고 있다.

[표 3] Kubernetes 스케줄러 우선순위

우선순위(Priorities)	설명
SelectorSpreadPriority	같은 Service, StatefulSet 또는 ReplicaSet에 속하는 Pod 수가 최소화되도록 분산
InterPodAffinityPriority	선호하는 Pod 간 친화성 및 반친화성으로 우선순위 지정
LeastRequestedPriority	요청된 리소스/Pod가 적은 노드를 선호
MostRequestedPriority	요청된 리소스/Pod가 많은 노드를 선호
RequestedToCapacityRatioPriority	리소스 스코어링 함수 셰이프를 사용하여 우선순위

우선순위(Priorities)	설명
BalancedResourceAllocation	자원 사용이 균형을 이루는 노드를 선호
NodePreferAvoidPodsPriority	두 개의 서로 다른 Pod가 동일한 Node에서 실행되는 것을 피하는 우선순위
NodeAffinityPriority	노드 친화성 및 반친화성으로 우선순위
TaintTolerationPriority	노드 내결함성 오염(taint)을 우선순위
ImageLocalityPriority	해당 포드의 컨테이너 이미지가 사전에 로컬로 캐시된 노드를 선호
ServiceSpreadingPriority	서비스를 위한 포드가 서로 다른 노드에서 실행되도록 하는 것을 우선순위
EqualPriority	모든 노드에 동일한 가중치를 부여
EvenPodsSpreadPriority	노드에 분산된 pod 수가 같도록 순위를 부여

〈자료〉 CNCF(Cloud Native Computing Foundation), “Priorities of Scheduling Policies on Kubernetes,” 2021.

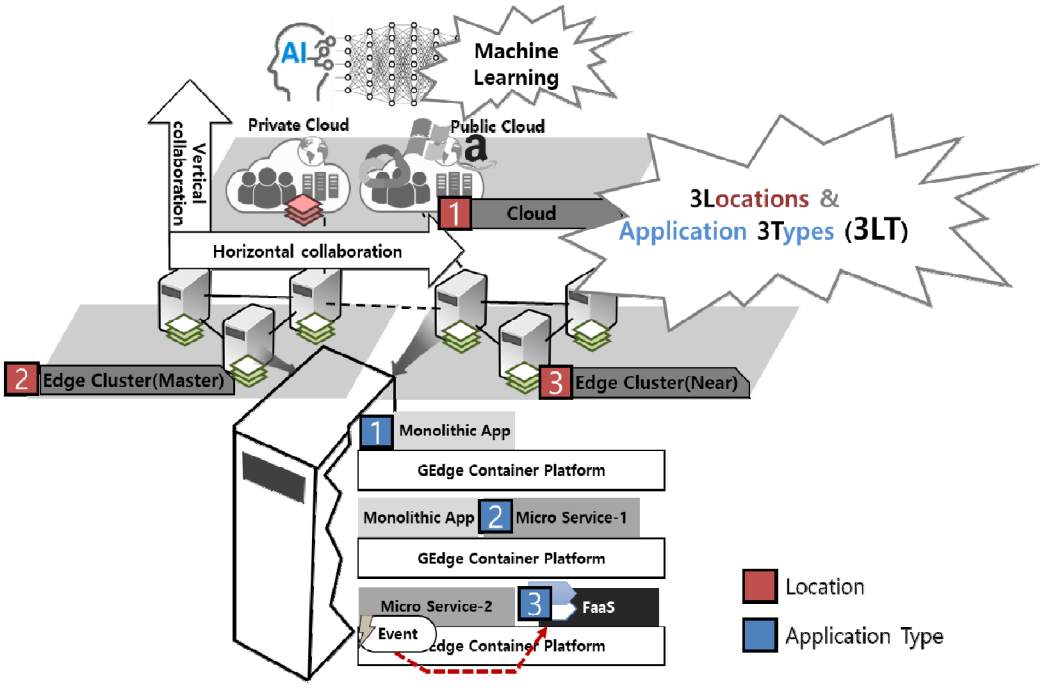
III. GEdge 플랫폼의 스케줄러

GEdge(Griffin Edge) 플랫폼은 엣지 컴퓨팅에 밀접하게 관련된 과제로 SW 컴퓨팅 산업 원천기술 개발 사업 과제인 “10msec 미만의 서비스 응답 속도를 보장하는 초저지연 지능형 엣지 SW 플랫폼 핵심 기술 개발” 과제에서 개발되고 있는 클라우드 중심의 엣지 컴퓨팅 플랫폼이다. GEdge 플랫폼은 응답 속도 민감형 서비스를 위해 클라우드-엣지-단말 간 협업 기반 초저지연 데이터 처리를 지원하는 지능형 엣지 SW 플랫폼 기술로 개발 진행되고 있다[8].

1. GEdge 플랫폼의 소개

기존 엣지 컴퓨팅과 비슷하지만 몇 가지 차별되는 특징을 가지고 있는 엣지 컴퓨팅 플랫폼인 GEdge 플랫폼은 [그림 3]과 같이 3LT(3 Locations and 3 Application Types)라는 서비스 실행환경 지원 특징을 가지고 있다.

기존 엣지 서비스를 위한 시스템 구성도가 기존 클라우드와 엣지 둘만의 관계가 있다면 엣지 컴퓨팅 서비스의 구성도에는 클라우드와 마스터 엣지 클러스터에 이웃 엣지 클러스터(Near Edge)로 구성된 3가지 위치(3 Locations)적 특성을 고려한 상호 협력을 기반으로 서비스가 이루어진다.



〈자료〉 한국전자통신연구원 자제작성

[그림 3] 엣지 컴퓨팅 서비스 실행 환경

- 클라우드(Public/Private Cloud)
- 마스터 엣지 클러스터(사용자 서비스를 주로 처리하는 엣지 클러스터)
- 이웃 엣지 클러스터(마스터 엣지 클러스터와 가까운 주변 엣지 클러스터)

[표 4]와 같이 마스터 엣지 클러스터는 네트워크 속도 측면에서 클라이언트와 바로 연결되어 있어서 지역별로 설치된 이웃 엣지 클러스터보다 빠르고, 사용할 수 있는 리소스 크기 면에서는 클라우드보다는 작은 규모이다. 인공지능 AI 트레이닝과 같이 많은 리소스를 요구하면서 빠른 응답 요청이 아닌 서비스 경우에는 클라우드에서 처리하는 것이 효과적일 것이다.

기존 엣지 서비스에서 실행되는 애플리케이션의 종류를 보면 기존 방식에서는 모놀리틱 응용과 마이크로 서비스 2가지 형태를 주로 처리하고 있다. 그러나 GEdge 플랫폼 기반의 엣지 서비스는 다음과 같이 서비스형 함수를 추가하여 3가지 애플리케이션 형태(3 Application Types)를 지원한다.

[표 4] 3 Locations 특징

구분	리소스 크기	네트워크 속도	서비스 처리 규모
마스터 엣지 클러스터	소(중)	상	소(중)
이웃 엣지 클러스터	소(중)	중	소(중)
클라우드	대	하	상

〈자료〉 한국전자통신연구원 자체 작성

- 모놀리틱 응용(Monolithic App)
- 마이크로 서비스(Micro Service)
- 서비스형 함수(FaaS)

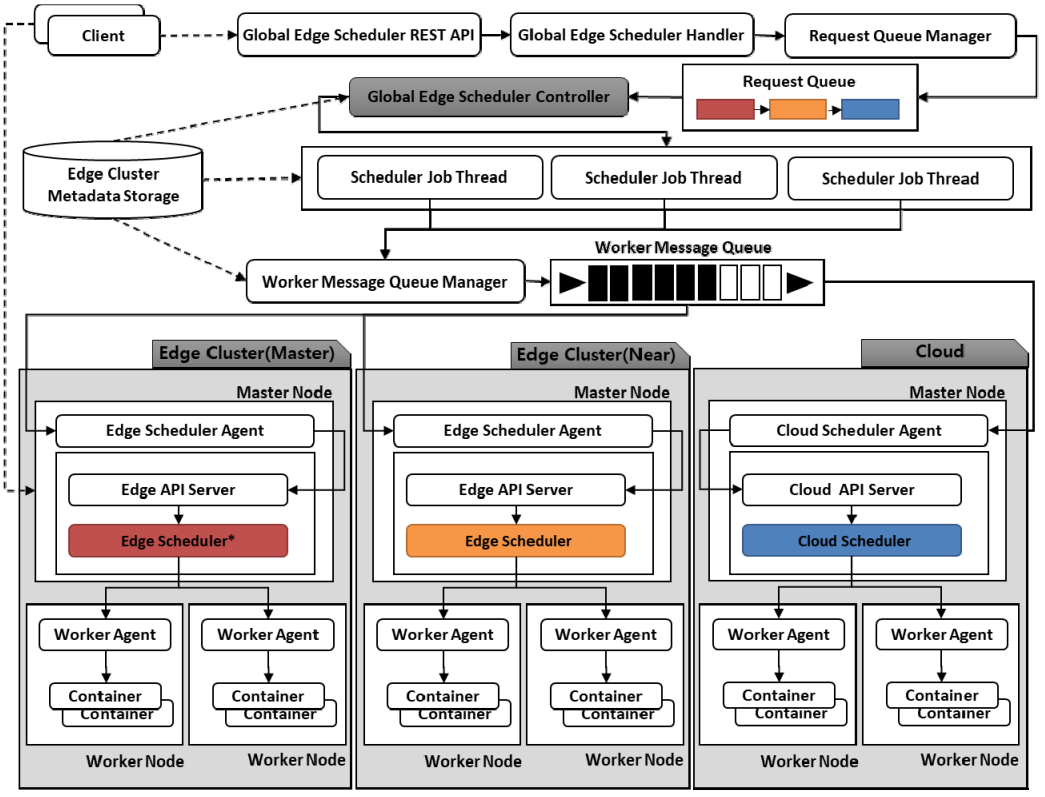
2. GEdge 플랫폼 스케줄러 구성도

GEdge 플랫폼의 스케줄러는 앞에서 언급한 3LT의 지역적 특성인 3 Locations를 담당하는 3가지 스케줄러와 통합적인 글로벌 스케줄링 처리를 위한 글로벌 엣지 스케줄러 제어기로 구성되어 있다. [그림 4]는 GEdge 플랫폼 스케줄러 구성도를 보여준다.

- 클라우드 스케줄러: Public/Private 클라우드 부분에서 작동되는 스케줄러
- 마스터 엣지 클러스터 스케줄러: 관리자가 서비스를 생성 요청할 때 지정되며 주된 서비스 할당이 이루어지는 엣지 클러스터의 스케줄러
- 이웃 엣지 클러스터 스케줄러: 마스터 엣지 클러스터 주변에 있는 엣지 클러스터들의 스케줄러로 서비스 이동이나 엣지 클러스터 간의 협업을 도와주는 역할을 한다.
- 글로벌 엣지 스케줄러 제어기: 클라우드/마스터 엣지 클러스터/이웃 엣지 클러스터 간에 서비스를 확대/연동하기 위한 글로벌 엣지 스케줄링을 제어하는 부분이다.

3. GEdge 스케줄러의 처리 메커니즘

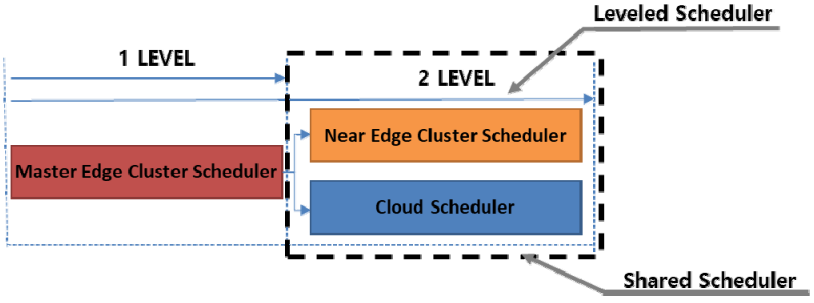
GEdge 스케줄러의 처리 메커니즘은 엣지 응용 서비스 간의 수평적/수직적 협업을 플랫폼 수준에서 적용하기 위해 설계되었다. GEdge 스케줄러는 순차 처리를 위한 Leveled 스케줄러 형식과 서로 경쟁하는 Shared 스케줄러 형식의 복합된 구조로 설계되었다. Leveled 스케줄러는 응용 개발자가 응용의 부분 서비스 부하를 이웃 엣지 클러스터나



〈자료〉 한국전자통신연구원 자체 작성

[그림 4] GEdge 플랫폼 스케줄러 구성도

클라우드로 분산시키거나 협업하는데 시스템 레벨에서 적용할 수 있어 매우 효과적이다. Shared 스케줄러는 경쟁 모드로 2개 이상의 스케줄링을 동시에 요청하여 각 스케줄러로



〈자료〉 한국전자통신연구원 자체 작성

[그림 5] GEdge 플랫폼 Leveled/Shared 스케줄러 예시

부터 제시된 후보 중에 가장 적합한 것을 선택하는 형태이다. [그림 5]의 예제는 1~2 Level로 구성된 Leveled 스케줄러이면서 2 Level에서 이웃 엣지 클러스터 스케줄러와 클라우드 스케줄러 간에 자원 할당에 대한 경쟁을 통해 자원이 할당되는 Shared 스케줄러로 구성된 복합적인 스케줄러를 보여준다.

제안된 GEdge 플랫폼의 스케줄러는 기존 컨테이너 오케스트레이션의 스케줄러 기술과 비교하여 아래와 같은 특성 및 우수성을 갖는다.

- 초저지연 서비스에 적합한 스케줄링 가능
- 수직적 수평적인 통합적인 협업 가능
- 시스템 레벨에서 협업 적용 가능
- 끊김이 없는 서비스 연계 가능
- 클라우드/엣지 클러스터/이웃 엣지 클러스터를 포함한 통합적인 분산처리 가능

Ⅳ. 결론

엣지 컴퓨팅을 위한 스케줄러는 클러스터의 자원들을 효율적으로 사용하고 사용자에게 제공되는 자원 배치의 제약 혹은 제한 기능이 작동되어야 하고 스케줄링 과정에서는 공정성을 가지고 요청한 작업을 신속하게 배치 및 할당하는 중요한 역할을 담당해야 한다.

기존의 대표 컨테이너 오케스트레이션 스케줄링 방식인 모놀리식 스케줄링, 2단계 스케줄링, 공유 상태 스케줄링은 각각의 특성이 있는데, 특히 공유 상태 스케줄링은 동시성 제어 측면에서 매우 효과적이고 모놀리식 스케줄링은 스케줄링의 연산 처리의 공정성에서 유리한 측면을 보여주고 있다. 엣지 컴퓨팅에서 가장 많이 사용되고 있는 Kubernetes의 스케줄러는 모든 리소스 할당 결정이 중앙 리소스 할당자가 없이 스케줄러에서 이루어지는 공유 상태 스케줄링 방식으로, 노드 필터링 과정과 노드 순위 지정 과정의 두 가지 과정을 통해 노드를 효과적으로 선정하고 있다. 그러나 Kubernetes의 스케줄러는 클라우드와 엣지 클러스터의 수직적 협력이나 엣지 클러스터 간의 수평적 협력과 같은 측면은 고려하고 있지 않다. 그래서 단일 클러스터의 스케줄러의 역할을 넘어서 다중 클러스터를 고려한 스케줄러로 기능을 확대할 필요가 있다.

GEdge 플랫폼의 스케줄러는 순차 처리를 위한 Leveled 스케줄러 형식과 서로 경쟁하

는 Shared 스케줄러 형식이 복합된 구조를 통해 다중 클러스터를 고려한 스케줄러로서, 응답 속도 민감형 서비스나 클라우드-엣지-단말 간 협업 기반 초저지연 데이터 처리를 지원하는 지능형 서비스를 효과적으로 지원할 수 있을 것으로 예측된다.

[참고문헌]

[1] Digital Today, "삼성전자-IBM, 5G 기반 엣지컴퓨팅 동맹", 2020.
[2] 홍정하, 이강찬, 이승윤, "엣지 컴퓨팅 기술 동향", 한국전자통신연구원, 전자통신동향분석, Vol.35 No.6, 2020, pp.78-87.
[3] 신성식 외, "엣지 컴퓨팅 시장 동향 및 산업별 적용 사례", 한국전자통신연구원, 전자통신동향분석, Vol.34, No.2, 2019, pp.51-59.
[4] 김영덕, "에지컴퓨팅(Edge Computing) 기술 동향과 시장분석", KOSEN 분석리포트, 2018.
[5] Jiasi Chen, Xukan Ran, "Deep Learning With Edge Computing: A Review," PROCEEDINGS OF THE IEEE Vol.107, No.8, Aug. 2019, pp.1655-1674.
[6] NVIDIA, "THE NVIDIA EGX AI PLATFORM," 2021.
[7] Victor Medel, Omer Rana, Jose Angel Banares, Unai Arronategui, "Adaptive Application Scheduling under Interference in Kubernetes," IEEE/ACM 9th International Conference on Utility and Cloud Computing, 2016, pp.426-427.
[8] 김선욱 외, "지능형 엣지 서비스를 위한 클라우드 엣지 SW 플랫폼 기술 동향", 정보와 통신:한국통신학회지, Vol.37, No.8, 2020, pp.46-54.
[9] Banzai Cloud, "Writing custom Kubernetes schedulers," 2019.