쿠버네티스에서의 DB 스케일링 기반 서비스 시간 개선 및 효율적인 자원 사용 방안

윤주녕 [1], 유헌창 [2]
[1] 고려대학교 SW·AI 융합대학원 석사과정
[2] 고려대학교 정보대학 컴퓨터학과 교수

nyoung9494@korea.ac.kr, yuhc@korea.ac.kr

A Study on Improved Service Time and Efficient Resource Utilization Based on DB Scaling in Kubernetes

Joonyoung Yoon[1], Heonchang Yu[2]
[1]Dept. of Software Security, Graduate School of SW·AI Convergence, Korea University
[2]Dept. of Computer Science & Engineering, Korea University

요 약

클라우드 사용이 보편화 되고 확대됨에 따라, 서비스를 유연하게 확장 및 축소하여 신속하게 시장의 수요에 대응할 수 있는 PaaS(Platform-as-a-Service) 형태의 서비스가 많은 기업에서 각광받고 있다. 그리고 이러한 PaaS 형 서비스의 핵심이 되는 기술인 컨테이너(Container)와 컨테이너 관리를 효율화 해주는 쿠버네티스(Kubernetes)가 실질적인 표준으로 사용되고 있다. 이때 쿠버네티스 기반의 환경에서 서비스 어플리케이션은 다양한 구성사례가 존재하나, DB 는 아직 안정성 및 데이터 정합성 등을 이유로 베어메탈(Baremetal)이나 VM(Virtual Machine)을 기반으로 구성하고 있는 상황이다. 그러나, 인프라 구성 및 운영에 있어서도 파드(Pod) 형태의 DB 구성은 베어메탈 및 VM 대비 장점이 존재한다고 생각하여 본 실험을 수행하였다. 본 논문에서는 서비스 응답시간 및 자원 사용의 효율성 측면에서 VM 기반의 DB 와 쿠버네티스 파드 기반의 DB 에 각각 트래픽을 발생시켜서 비교한 결과와 시사점을 제시한다.

1. 서론

대부분의 기업에서 추구하는 가치가 비용 효율화 및 Time-to-Market 최소화임에 따라, 클라우드 사용이 더욱 확대되고 있다. 초기 클라우드 서비스 제공방식은 기본적인 하드웨어 자원만 제공하는 IaaS(Infra-as-a-Service) 방식이었지만, 점차 클라우드에 대한 수요가 증가하고 서비스가 고도화됨에 따라 OS 부터 Middleware 및 어플리케이션 런타임까지 제공하는 PaaS(Platform-as-a-Service)가 보편화되었다.

PaaS 형 서비스 환경에서 소프트웨어는 파드(Pod) 형태로 배포 및 구동되는데, 이론상으로 파드는 1 개 이상의 컨테이너(Container)로 구성될 수 있다. 그러나, 일반적으로는 자원의 격리나 서비스의 독립적인 운영 측면에서 1 개의 파드는 1 개의 컨테이너로 구성한다.

컨테이너는 사전에 정의해 둔 특정한 소프트웨어의 형상으로 볼 수 있는데, 이러한 형상은 이미지(Image)라는 형태로 관리되고 사용자는 도커허브(Docker Hub)와 같은 이미지 저장소로부터 이미지를 불러와 소프트웨어를 파드 형태로 실행한다.

파드의 개수가 적다면 사람이 직접 파드의 생성과 폐기부터 개수 조정과 같은 세부사항까지 처리할 수 있으나, 복잡한 서비스를 제공하는 기업 운영 환경에서는 수백개에서 수천개의 파드가 실행되어 사람이 직접 관리하기에는 상당한 어려움이 존재한다.

이를 위해 쿠버네티스(Kubernetes)라는 기술이 등장하여 파드 관리를 위해 수행해야 하는 많은 작업을 자동화해주었다. 쿠버네티스는 여러 개의 서버 혹은 노드를 하나의 클러스터로 구성하여 다양한 소프트웨어가 물리적인 서버에 대한 종속없이 유연하게 실행될 수 있도록 하는데, 사용자 입장에서는 소프트웨어가 클러스터 내의 어느 노드에서 실행되는지 알 필요없이 서비스만 운영하면 된다.

쿠버네티스 기반으로 운영되는 서비스 어플리케이션은 보편화되었고 많은 구성사례가 존재하나, DBMS

를 쿠버네티스에 설치하여 운영하는 사례는 찾아보기 힘들다. 이는 파드의 운영을 쿠버네티스가 관장함에 따라 임의적인 파드 삭제에 의한 데이터 유실 및 정합성 측면의 우려가 존재하기 때문이다. 그러나, Persistent Volume 을 통해 해당 문제를 해소할 수 있고, 더 나아가 파드 형태의 DB 구성을 통해 인프라 운영과 자원 활용 측면에서도 효율성 향상을 기대할 수 있다.

따라서, 본 논문에서는 VM 기반의 DB 와 쿠버네티스 기반의 DB 를 각각 구성하고 스케일 업(Scale-up)이 가능하도록 하여, 데이터 처리 요청에 대한 두 DB 구성방식 간의 서비스 응답시간 및 자원 사용 효율성을 비교하고자 한다. 실험 결과, 쿠버네티스 기반의 DB 에 오토스케일링(Autoscaling) 정책을 적용했을 때 서비스 응답시간 및 DB 자원 최적화 측면에서 VM 기반의 DB 대비 유의미한 이점이 있는 것으로 확인되었다.

2. 관련 연구

2.1 쿠버네티스 오토스케일링의 성능에 관한 연구

많은 연구에서 어플리케이션 수준의 HPA(Horizontal Pod Autoscaling) 및 VPA(Vertical Pod Autoscaling) 적용을 실험하였고, 실제로 응답시간이나 처리량 측면에서 HPA 의 장점을 보여주기도 하였다. [1]

특히, 두 방식을 혼합한 Hybrid 형태로 어플리케이션의 성능을 연구한 사례도 있으나 [2], 쿠버네티스 DB 에 특화된 오토스케일링 연구는 다소 부족한 실정이다. 본 연구에서는 쿠버네티스 DB 에 특화된 인프라를 구성하여 어플리케이션이 아닌 DB 수준에서도 오토스케일링 적용이 가능할지 실험하고 관련 지표들을 제시하고자 하였다.

3. 쿠버네티스 기반의 DB 서버 구성

대다수의 기업에서 DB 서버를 구성하는 방식은 예상되는 Transaction 의 양과 관련된 DB I/O 빈도를 계산하여 물리적인 서버(베어메탈 혹은 VM)의 개수와 서버별 자원을 사전에 정의하는 방식이다. 이러한 방식의 단점은 서버 구성 형태를 유연하게 변경하기 어려운 점과 서버에 할당된 자원을 쉽게 조정하기 어렵다는 점이다. 즉, DB 서버의 안정성 향상을 위해 시스템 구성 형태를 변경해야 하거나 불필요한 자원의 낭비가 있어서 개선이 필요해도 빠르게 대응하기 어렵다.

반면, 쿠버네티스로 서버를 구성하는 경우에는 사전에 구성된 클러스터 내에서 파드 단위로 서버 구성 형태나 자원을 유연하게 조정할 수 있다. 특히, DB 서버의 구조는 크게 쿼리(Query)에 대한 논리적인 처리를 진행하는 소프트웨어 영역(DBMS)과 실제 데이터를 저장/반영하는 하드웨어 영역(디스크/스토리지)으로 구분할 수 있는데, 쿼리를 처리하기 위해 자원을 많이 요구하는 소프트웨어 영역을 파드로 구성하면 DB 성능의 향상을 기대할 수 있다.

베어메탈 및 VM DB 구성과 비교하여 DB 를 쿠버네티스 상에 파드 형태로 구성할 때의 주된 이점은 다음과 같다.

1) 자원 사용률 효율화
2) HA(High Availability) 구성 용이성
3) 트래픽 증감에 따른 서비스 탄력성 향상

3.1 자원 사용률 효율화

베어메탈 및 VM 의 경우에는 한 번 설정한 자원 할당량을 조정하기 쉽지 않다. 물론, VM 의 경우에는 VM 도구에서 제공하는 자원 할당량 관리 기능이 있으나, VM 재부팅에 따른 서비스 영향도를 사용자가 직접 충분히 고려해야 한다. 또한, 디스크의 경우에도 일반적으로는 용량 증설 기능만을 제공하므로 처음 DB 구성 시 과도하게 용량이 설정된 경우에는 별도의 데이터 이관 방안을 수립하여 용량 축소를 진행해야 하는 불편함이 존재한다.

반면, 파드 형태로 구성된 DB 는 쿠버네티스가 파드의 생성 및 삭제를 관리해주므로 사용자 입장에서는 베어메탈 및 VM 방식 대비 편하게 자원 할당량을 조절할 수 있다.

예를 들어, 파드를 쿠버네티스 환경에 배포하기 위한 deployment.yaml 파일 내에 파드별로 할당할 자원의 크기를 명시하고, 운영 과정에서 자원이 부족하거나 과도하게 할당된 경우에는 해당 yaml 파일을 수정하여 간단하게 파드의 자원 할당량을 조절할 수 있다.

3.2 HA 구성 용이성

베어메탈 및 VM 은 HA 구성을 위해 별도의 클러스터링 작업과 데이터 복제/공유 환경 구성을 진행해야 한다. 일례로, 본 연구에서는 2 대의 DB 노드(VM)에 대해 노드별 Healtcheck 를 위한 keepalived 설정과 데이터 공유를 위한 DRBD(Distributed Replicated Block Device) 구성을 진행하여 HA 가 가능하도록 하였다.

이에 비해 파드 형태의 DB 는 별도의 소프트웨어를 설치해야 하는 번거로움이 적다. 쿠버네티스 클러스터 내부에서 deployment 를 선언하여 생성된 파드 간에는 자동으로 로드밸런싱이 이루어지기 때문에, 사용자가 별도로 클러스터링을 할 필요가 없다. 또한, 데이터 복제/공유를 위해서는 간단한 yaml 파일의 작성 및 적용을 통해 Persistent Volume 을 설정하여 DB

파드가 동일한 데이터를 참조할 수 있도록 구성하면 되므로 기타 소프트웨어 설치 방식에 비해 매우 간편하다.

3.3 트래픽 증감에 따른 서비스 탄력성 향상

베어메탈의 경우에는 메모리를 메인보드 슬롯에 추가적으로 장착하는 방식으로 스케일 업을 할 수 있고, VM 의 경우에는 VM 도구에서 자원을 재조정하는 방식으로 스케일 업을 할 수 있다. 그러나, 베어메탈 및 VM 환경에서의 DB 스케일 아웃(Scale-out)을 위해서는 클러스터링을 위한 별도의 소프트웨어 설치나 부수적인 작업이 요구되고, 데이터 복제와 공유 방안에 대해서도 면밀히 검토하고 진행해야 한다.

반면, 쿠버네티스 기반으로 DB 를 파드 형태로 구성한 경우에는 앞서 언급한 Persistent Volume 을 활용했다고 가정할 때, 쿠버네티스에서 제공하는 오토스케일링 기능을 통해 간단하게 스케일 업과 스케일 아웃을 진행할 수 있다. 특히, HPA 및 VPA 정책을 yaml 파일 하나로 설정할 수 있으므로 베어메탈이나 VM 대비 손쉽게 스케일 업 및 스케일 아웃을 설정할 수 있다.

4. 쿠버네티스 기반의 DB 서버 Scaling 실험

4.1 호스트 서버 구성

별도의 OS 가 설치되어 있지 않은 일반 PC 에 베어메탈 Hypervisor 인 Proxmox 를 설치하여 Hybrid Hypervisor 대비 Host OS 에 의한 영향도를 최소화하고, 해당 환경 내에서 VM 을 생성하였다.

4.2 VM 기반 DB 영역 구성

2 대의 VM 각각에 MySQL 8.0 을 설치하였으며, keepalived 소프트웨어를 통해 두 VM 을 Virtual IP 로 클러스터링하여 액티브-스탠바이(Active-Standby) 구성이 되도록 하였다. keepalived service 에 의해 각 VM 의 백그라운드 프로세스에서 Healthcheck 가 이루어지며, 액티브 노드에서 장애 발생 시 스탠바이 노드로 Virtual IP 가 자동으로 바인딩되어 DB Connection 이 중단되지 않을 수 있다.

DB 운영 측면에서 스탠바이 노드로 트래픽 전환(Failover)이 이루어져도 어플리케이션이 참조하는 데이터는 동일해야 하므로, DRBD(Distributed Replicated Block Device)를 구성하여 데이터 정합성 문제를 해소하였다. 이를 위해 각 VM 에 별도 파티션을 구성하고 공유 볼륨으로 활용하여 Failover 시 액티브 노드의 MySQL 데이터 저장 디렉터리에 자동으로 마운트될 수 있도록 하였다.

DB 스케일 업과 동일한 환경을 구성하기 위해 액티브 노드보다 스탠바이 노드에 자원을 더 많이 할당하였다(표 1).

<표 1> VM DB 노드별 자원 할당량

구분	액티브 노드	스탠바이 노드
vCPU	3	6
Memory	4 GiB	6 GiB

4.3 쿠버네티스 DB 영역 구성

1 대의 VM 에 로컬 쿠버네티스인 KinD(Kubernetes in Docker)를 설치하여 환경을 구성하였다. VM 과 마찬가지로 MySQL 8.0 버전을 설치하였으나, deployment 방식으로 파드를 생성하였다. 이때 VM 과 유사한 형태로 구성하기 위해 파드는 1 개만 생성하도록 제한하였다.

VPA 정책 적용을 통해 MySQL 의 자원 사용률에 따라 스케일 업된 파드가 새롭게 생성되고, 기존 파드는 삭제되도록 설정하였다. VM DB 와의 적절한 비교를 위해 파드에 대한 자원 할당 범위를 사전에 정의하였는데(표 2), 앞서 구성한 VM DB 가 유휴 상태에서의 Memory 사용량이 2 GiB 였으므로 이를 고려하여 VM 대비 Memory 를 2 GiB 씩 적게 할당하였다. CPU 는 vCPU 형태이므로 VM 과 동일하게 구성하였다.

<표 2> DB 파드 자원 할당량

구분	Min	Max
vCPU	3	6
Memory	2 GiB	4 GiB

Persistent Volume 을 설정하여 파드가 변경되더라도 호스트 VM 의 특정 디렉터리에 MySQL 데이터가 적재되어 데이터 정합성 문제가 없도록 구성하였고, MySQL 파드에 대해 쿠버네티스 NodePort 서비스를 활용하여 쿠버네티스 클러스터 외부에서 DB 에 접속할 수 있도록 하였다.

5. 실험결과

각 DB 서버의 자원 사용률에 대한 외부의 영향을 최소화하기 위해 호스트 서버에 테스트 부하 발생을 위한 VM 을 별도로 구성하였다. 해당 VM 내에 테스트 도구인 Jmeter 를 설치하고 각 DB 서버와 연결하여 직접 쿼리를 보내는 방식으로 부하를 발생시켰다.

Jmeter 테스트 설정값 중 DB 동시 접속자 수로 볼 수 있는 Number of Threads 값은 151 로 설정하였으며, 접속자별 호출횟수인 Loop Count 는 10,000 으로 하여 총 1,510,000 회의 쿼리가 발생되도록 하였다. 본 실험에서는 100 건의 레코드에 대해 Select 쿼리를 실행하여 여러 성능지표에 대한 측정을 진행하였다.

5.1 DRBD 구성을 통한 스케일 업 실험결과

DRBD 구성 내 액티브 노드를 강제로 셧 다운하여 더 많은 자원을 보유한 스탠바이 노드로 트래픽을 전환시킨 후 평균응답시간, 수행시간, 에러율을 측정하였다. 평균응답시간은 각 호출별 응답시간의 평균값이며, 수행시간은 호출에 따른 응답이 전부 완료되기까지 소요된 시간이다. 에러율은 각 호출에 대해 DB가 처리 후 응답한 결과를 토대로 산정하며, 비정상적인 응답이 있을 시 에러율이 증가하게 된다.

스케일 업 적용을 위해 Jmeter 에서 테스트 실행 후 즉시 액티브 노드를 직접 셧 다운하였다. 스케일 업을 적용하지 않은 경우 대비 수행시간이 평균 약 11초 정도 줄었고, 이는 서비스 운영 측면에서 유의미한 수치라고 볼 수 있다. 다만, 에러율이 확연하게 상승하였는데, 이는 스탠바이 노드로 전환하고 디스크를 마운트하는 동안 DB 가 제대로 응답을 줄 수 없었기 때문이다.

<표 3> VM 기반 DB 의 스케일 업 성능측정

구분	스케일 업 미실행	1 차	2 차	3 차
평균응답시간	8ms	6ms	6ms	6ms
수행시간	97s	84s	84s	88s
에러율	0.00%	9.24%	7.45%	8.19%

5.2 VPA 구성을 통한 스케일 업 실험결과

VPA 적용의 효과를 확인하기 위해 VPA 적용 전 Deployment.yaml 내 자원 limit 을 제한한 상태에서 실험을 먼저 진행하였다. DB 파드가 사용할 수 있는 자원의 최대치를 3000m(vCPU), 2048Mi(메모리)로 고정하고 앞선 VM DB 실험과 동일하게 쿼리를 실행했다.

이후 VPA 를 실행한 상태에서 VPA 가 DB 파드에 대한 최적의 자원 할당 값을 찾을 수 있도록 동일한 쿼리를 2 번 더 실행하였고, 3 번째 실행과정에서 VPA 가 동작하였다. VPA 가 동작하면서 새로운 파드가 생성되고 더 많은 자원을 사용할 수 있게 됨에 따라, 평균응답시간과 수행시간이 모두 감소한 것을 확인하였다. 다만, VPA 가 동작에 따른 파드 재생성 시간 동안 쿼리에 대해 응답을 줄 수 없게 되어 에러율이 증가하였다.

마지막으로, VPA 가 최적의 자원 값을 할당한 파드가 유지되고 있는 상태에서 쿼리를 실행한 결과, 앞선 두 상태 대비 성능 향상이 있음을 확인할 수 있었고, 특히 자원 할당 값을 고정한 상태 대비 평균응답시간과 수행시간이 절반 이상 감소하였다.

<표 4> VPA 기반 DB 의 스케일 업 성능측정

구분	자원 고정	VPA 동작	VPA 완료
vCPU 할당량	최대 3000 m	최대 3481m	최대 3481m
평균응답시간	10ms	7ms	4ms
수행시간	111s	74s	52s
에러율	0.00%	6.39%	0.00%

※메모리의 경우 사용률 변동폭이 크지 않았음(450Mi 내외에서 변동)

6. 결론

Select 쿼리를 중심으로 스케일 업에 따른 성능을 측정한 결과, 전반적인 수행시간에 있어서 유의미한 효과가 있었다. 비록 Failover 및 파드 재생성 시간 동안 적절한 응답이 불가하여 약간의 에러가 존재하였지만, 조회서비스만을 제공하는 DB 의 경우에는 충분히 적용을 고려해볼 수 있을 수준으로 사료된다.

DRBD 구성을 통한 스케일 업 실험의 경우, 전체 수행시간 측면에서 유의미한 결과가 있었으나 에러율 상승이라는 단점이 분명하게 존재하였다. 이를 극복하기 위해 DRBD 외의 다른 방식으로 Failover 시간을 단축할 수 있는 방안을 실험하거나, 어플리케이션 수준에서 DB 의 비정상 응답을 적절하게 처리할 수 있도록 프로그램을 구현하는 방안을 실험해 볼 필요가 있다고 판단된다.

VPA 적용을 통한 스케일 업 실험의 경우, 미적용 시에는 VM 기반의 DB 보다 성능이 낮았지만, VPA 가 최적의 자원 할당 값을 찾은 후부터는 VM 기반의 DB 보다 전반적인 성능이 향상된 것을 확인할 수 있었다. 또한, 에러율을 통해 볼 때 VM 의 Failover 시간보다 파드의 재생성 시간이 더 짧아서 서비스 다운타임 최소화 측면에서도 효과적일 것으로 판단된다. 추가적으로, 몇 번의 동일한 쿼리 실행을 통해 DB 에 필요한 최적의 자원을 찾아낼 수 있었으므로 자원 사용 측면에서도 파드 형태의 DB 구성이 보다 유리한 것으로 판단된다.

본 실험에서는 Select 쿼리만을 수행하였으나, 추후 실험에서는 Insert 및 Update 쿼리도 수행하여 데이터 정합성 측면에서도 두 DB 구성방식을 비교하고자 한다. 또한, DB 스케일 업 시간 동안의 에러율을 최소화하기 위해 DB 파드 자체에 Graceful Termination 정책을 적용하거나, 어플리케이션 수준에서 Graceful 셧 다운 정책을 적용하는 방안도 고려해보고자 한다.

참고문헌

[1] P. Choonhaklai and C. Chantrapornchai, "Two Autoscaling Approaches on Kubernetes Clusters Against Data Streaming Applications," 2023 International Technical Conference on Circuits/Systems, Computers, and Communications (ITC-CSCC), Jeju, Korea, Republic of, 2023, pp. 1-6

[2] Do, Truong-Xuan, and Vu Khanh Ngo Tan. "Hybrid Autoscaling Strategy on Container-Based Cloud Platform." IJSI vol.10, no.1 2022: pp.1-12.