aws

급격한 트래픽 증가를 처리하기 위한 하이퍼스케일링 Aurora MySQL 호환

AWS 권장 가이드

AWS 권장 가이드: 급격한 트래픽 증가를 처리하기 위한 하이퍼스케일링 Aurora MySQL 호환

Table of Contents

소개 1
목표 비즈니스 성과 1
연결 관리 3
구성 변수 3
연결 풀링 구현 4
쿼리 워크로드 조정 6
워크로드에서 문제가 되는 쿼리 추적 및 조정 6
쿼리 조정 가이드 8
스캔한 행 수 최소화 8
임시 테이블 사용량 및 디스크의 임시 테이블 최소화 8
파일 정렬 방지 9
동시성이 높은 집계 쿼리 실행 방지 9
쿼리의 동시성 테스트 9
쓰기 워크로드 조정 9
참조 무결성 이동 10
무거운 프라이머리 키 피하기 10
파티션 교환 사용 10
사용되지 않는 인덱스 삭제 10
이전 행 버전 제거 10
로깅 11
부하 제한 11
읽기 및 쓰기 워크로드 분리 12
오래된 데이터 보관 또는 삭제 12
중요하지 않은 ETL 프로세스 연기 13
애플리케이션 기능 끄기 13
파라미터 튜닝 15
리소스 16
문서 기록 17
..... xviii

급격한 트래픽 증가를 처리하기 위한 하이퍼스케일링 Aurora MySQL 호환

Oliver Francis, Shyam Sunder Rakhecha 및 Vikram singh Rai, Amazon Web Services(AWS)

2022년 10월

인터넷상의 귀하의 비즈니스는 기존 애플리케이션 인프라가 처리할 수 없는 전례 없는 고성장에 직면할 수 있습니다. 이는 제품에 대한 광고가 예상보다 많은 관심을 불러일으키거나 고객 구매 패턴이 대면 방식에서 온라인으로 갑자기 바뀌는 등 다양한 요인에 따라 발생할 수 있습니다.

이러한 예상치 못한 부하를 해결하려면 인프라를 하이퍼스케일링해야 합니다. 애플리케이션 티어를 확장하려면 서버나 포드를 더 추가해야 합니다. 그러나 하이퍼스케일링 수행에 있어 가장 어려운 부분은 데이터베이스입니다. 데이터베이스 인스턴스를 사용 가능한 최대 인스턴스 크기로 확장할 수 있지만 이렇게 해도 문제가 해결되지 않을 수 있습니다.

Amazon Aurora MySQL 호환 버전에서 데이터베이스 워크로드를 실행하는 경우 이 가이드는 데이터베이스를 하이퍼스케일링하여 고성장에 맞춰 구현할 수 있는 권장 사항을 제공합니다. 이러한 권장 사항 모두가 장기적인 모범 사례는 아닙니다. 고성장을 예상하고 이를 해결할 계획을 세울 시간이 있다면 리소스 섹션에 나열된 블로그 게시물을 참조하세요. 이러한 게시물은 장기적인 모범 사례를 구현하는 데 도움이 될 수 있습니다. 반면, 예상치 못한 고성장에 직면한 경우 이 가이드는 비즈니스를 위한 장기 솔루션을 계획하고 구현하는 동시에 부하를 관리하고 합리적인 안정성을 확보하는 데 도움이 될 것입니다.

이 가이드에 설명된 권장 사항에는 개발 팀의 구현 지원이 필요하다는 점에 유의하세요.

목표 비즈니스 성과

이 가이드에서 다루는 접근 방식은 다음에 도움이 됩니다.

- 예상치 못한 고성장 상황에서 비즈니스 안정화 고성장을 위한 장기적 모범 사례를 구현할 수 있을 만큼 충분한 안정성 확보

- 재정적 손실 방지 하이퍼스케일된 환경에서 갑작스러운 중단이 발생하면 고객이 애플리케이션에서 수행하는 비즈니스 트랜잭션이 감소할 수 있습니다. 이로 인해 경우에 따라 상당한 재정적 손실이 발

생할 수 있습니다. 안정화된 하이퍼스케일된 환경은 비즈니스 손실로 이어지는 장기간의 운영 중단을 방지하는 데 매우 중요합니다.

연결 관리

애플리케이션에 대한 수요가 증가하면 프런트엔드 트래픽도 증가합니다. 일반적인 시나리오에서는 이러한 수신 트래픽 폭증을 처리하기 위해 애플리케이션 계층에서 자동 크기 조정을 설정합니다. 그 결과 애플리케이션 티어 크기가 자동으로 조정되기 시작하고 트래픽 증가에 맞춰 더 많은 애플리케이션 서버(인스턴스)가 추가됩니다. 모든 애플리케이션 서버에는 데이터베이스 연결 풀 설정이 사전 구성되어 있기 때문에 데이터베이스의 수신 연결 수는 새로 배포된 인스턴스에 비례하여 증가합니다.

예를 들어, 애플리케이션 서버 20개를 각각 200개의 데이터베이스 연결로 구성하면 총 4,000개의 데이터베이스 연결이 열립니다. 애플리케이션 풀이 최대 200개 인스턴스까지 스케일 업되는 경우(예: 사용량이 많은 시간) 총 연결 수는 40,000개에 달합니다. 일반적인 워크로드에서는 이러한 연결 대부분이 유휴 상태일 가능성이 높습니다. 그러나 연결이 급증하면 Amazon Aurora MySQL 호환 버전 데이터베이스의 크기 조정 능력이 제한될 수 있습니다. 이는 유휴 연결이라도 메모리와 파일 설명자 등의 기타 서버 리소스를 소비하기 때문입니다. Aurora MySQL 호환은 일반적으로 동일한 수의 연결을 유지하기 위해 MySQL Community Edition보다 적은 메모리를 사용합니다. 그러나 유휴 연결의 메모리 사용량은 여전히 0이 아닙니다.

구성 변수

`max_connections`와 `max_connect_errors`라는 두 가지 주요 서버 구성 변수를 사용하여 데이터베이스에 허용되는 수신 연결 수를 제어할 수 있습니다.

구성 변수 max_connections

구성 변수 `max_connections`는 각 MySQL 인스턴스의 데이터베이스 연결 수를 제한합니다. 가장 좋은 방법은 각 데이터베이스 인스턴스에서 열 것으로 예상되는 최대 연결 수보다 약간 높게 설정하는 것입니다.

`performance_schema`도 활성화한 경우 `max_connections` 설정에 특히 주의하세요. 성능 스키마 메모리 구조는 `max_connections`를 포함한 서버 구성 변수에 따라 자동으로 크기가 조정됩니다. 변수를 높게 설정할수록 성능 스키마가 사용하는 메모리가 많아집니다. 극단적인 경우에는 크기가 작은 인스턴스 유형에서 메모리 부족 문제가 발생할 수 있습니다. Performance Insights를 활성화하면 Performance Schema가 자동으로 활성화됩니다.

구성 변수 max_connect_errors

구성 변수 `max_connect_errors`는 특정 클라이언트 호스트에서 허용되는 연속적인 중단 연결 요청 수를 결정합니다. 클라이언트 호스트의 연속 연결 실패 횟수가 지정된 횟수를 초과할 경우 서버는 해당 연결을 차단합니다. 해당 클라이언트에서 연결을 계속 시도하면 오류가 발생합니다.

```
Host 'host_name' is blocked because of many connection errors. Unblock with
'mysqladmin flush-hosts'
```

"호스트가 차단됨" 오류가 발생하는 경우 `max_connect_errors` 변수 값을 늘리지 마십시오. 대신 `aborted_connects` 상태 변수와 `host_cache` 테이블에서 서버의 진단 카운터를 조사합니다. 수집된 정보를 사용하여 연결 문제가 발생하는 클라이언트를 식별하고 수정합니다. 또한 `skip_name_resolve`가 1(기본값)로 설정된 경우 이 파라미터는 아무런 효과가 없습니다.

다음에 대한 자세한 내용은 MySQL 참조 설명서를 참조하세요.

- Max_connect_errors 변수
- "호스트가 차단됨" 오류
- Aborted_connects 상태 변수
- Host_cache 테이블

연결 풀링 구현

규모 조정 이벤트로 인해 애플리케이션 서버가 더 추가될 수 있으며, 이로 인해 DB 서버가 완전히 로드된 활성 연결 수를 초과할 수 있습니다. 애플리케이션 서버와 데이터베이스 사이에 연결 풀이나 프록시 계층을 추가하면 깔때기 역할을 하여 데이터베이스의 총 연결 수가 줄어듭니다. 프록시의 주요 목적은 멀티플렉싱을 통해 데이터베이스 연결을 재사용하는 것입니다.

한쪽에서는 프록시가 제어된 수의 연결을 사용하여 데이터베이스에 연결됩니다. 다른 쪽에서는 프록시가 애플리케이션 연결을 수락합니다. 또한 쿼리 캐싱, 연결 버퍼링, 쿼리 다시 작성 및 라우팅, 로드 밸런싱과 같은 추가 기능을 제공합니다. 데이터베이스에 대한 최대 연결 수를 완전히 로드된 수 이하로 유지하도록 연결 풀 계층을 구성해야 합니다. Amazon RDS 프록시는 이러한 목적으로 구현할 수 있는 완전관리형 프록시입니다. Amazon RDS 프록시는 대부분의 애플리케이션에서 코드를 변경할 필요가 없으며 솔루션을 구현하기 위해 추가 인프라를 관리할 필요가 없습니다.

Aurora MySQL 호환과 함께 사용할 수 있는 다음과 같은 타사 프록시를 탐색할 수도 있습니다.

- ProxySQL
- MariaDB MaxScale
- ScaleARC
- Heimdall Data

연결 폭풍 피하기

데이터베이스가 과부하되거나 복제본이 기본 노드보다 너무 뒤처지는 경우 연결 풀이 어떻게 작동하는지 고려하세요. 프록시 서버 또는 연결 풀을 구성할 때 기본 하드웨어 또는 스토리지 문제 또는 DB 리소스 제약으로 인한 느린 데이터베이스 응답을 기준으로 전체 연결 풀을 재설정하지 않도록 합니다.

갑자기 수백 개의 연결을 시작하면 데이터베이스에 대한 많은 수의 새 연결 요청이 동시에 시작되므로 연결 폭풍이 발생합니다. 폭풍은 리소스를 많이 소모합니다. MySQL에서 새 데이터베이스 연결을 만드는 것은 백엔드가 초기 핸드셰이크를 위해 여러 네트워크 패킷을 교환하고, 새 프로세스를 생성하고, 메모리를 할당하고, 인증을 처리하는 등의 작업을 수행하기 때문에 비용이 많이 드는 작업입니다. 짧은 시간 내에 많은 요청을 받으면 데이터베이스가 응답하지 않는 것처럼 보일 수 있습니다.

MySQL에는 이러한 연결 요청 급증을 방지하는 메커니즘이 있습니다. `back_log` 변수는 MySQL이 새 요청에 대한 응답을 잠시 중단하기 전에 잠시 동안 누적될 수 있는 요청 수로 설정할 수 있습니다. 이 값은 연결 처리 스레드에 의해 적용되며, 이 스레드 자체가 연결 폭풍으로 인해 과부하될 수 있습니다. 자세한 내용은 MySQL 참조 매뉴얼을 참조하세요.

데이터베이스 속도가 느릴 때 연결이 재설정되도록 구성된 경우 주기가 반복해서 시작됩니다. 마찬가지로, 하루 중 특정 시간(예: 주식 시장이 열리는 시간)에 데이터베이스 트래픽이 갑자기 증가할 것으로 예상되는 경우, 높은 트래픽 부하가 시작되는 동시에 많은 연결을 열려고 하지 않도록 연결 풀을 예열합니다.

쿼리 워크로드 조정

워크로드를 잘 조정하면 고성장을 위한 안정적인 솔루션을 구축하는 데 많은 도움이 됩니다. 워크로드가 제대로 조정되지 않으면 사용하는 Amazon Aurora 클러스터의 성능과 상관없이 성능이 저하되고 애플리케이션의 사용자 경험에 영향을 미치는 병목 현상이 발생합니다. 가장 좋은 방법은 시스템에서 문제가 되는 쿼리를 식별하고 조정하는 데 도움이 되는 프로세스를 처음부터 마련하는 것입니다.

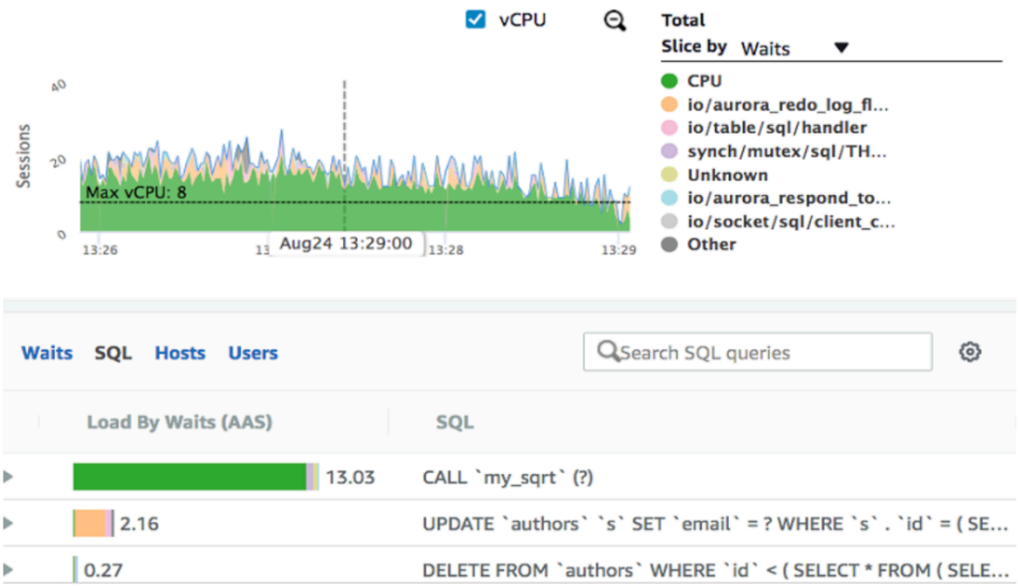
워크로드에서 문제가 되는 쿼리 추적 및 조정

고성장에 직면했을 때 워크로드를 잘 조정하면 절반은 이긴 게임입니다. Aurora 클러스터가 직면하고 있는 실시간 워크로드 및 성능 문제의 특성을 이해하려면 팀이 작성자 인스턴스와 리더 인스턴스 모두에서 문제가 되는 쿼리를 수집해야 합니다. 워크로드가 최적 상태에서 실행되도록 이러한 문제가 있는 쿼리를 조정해야 합니다. Amazon Aurora MySQL 호환 버전은 이 작업을 수행하는 두 가지 방법을 제공합니다.

- 성능 개선 도우미
- 느린 쿼리 로그

성능 개선 도우미 사용

성능 개선 도우미는 평균 활성 세션(AAS)을 기준으로 Aurora 라이터 또는 리더 인스턴스의 부하를 추적합니다. AAS 값은 샘플링과 CPU가 쿼리 워크로드를 선택하고 처리하기를 기다리는 활성 세션 수를 사용하여 계산됩니다. 성능 개선 도우미는 활성 세션 대기로 인한 최대 부하를 유발하는 SQL 문을 확인할 수 있는 그래픽 인터페이스를 제공합니다.



이전 스크린샷에서는 저장 프로시저 `my_sqrt`에 대한 호출로 인해 평균 13.03개의 세션이 부하 처리를 기다립니다. 논리적인 다음 단계는 이 절차를 조정하는 것입니다. 리더와 라이터에서 해당 인스턴스에 부하를 유발하는 SQL 문을 식별하고 AAS 값이 떨어지고 성능 개선 도우미에서 최대 vCPU 점선 아래로 유지될 때까지 성능을 개선하도록 SQL 문을 조정해야 합니다. 조정 작업이 한도에 도달했는데도 여전히 AAS가 최대 vCPU 제품군보다 높다면 더 큰 인스턴스 클래스를 선택하여 워크로드를 처리할 수 있습니다. 먼저 쿼리 워크로드를 조정하지 않고는 더 큰 인스턴스를 선택하지 마세요. 트래픽이 증가하면 워크로드의 잘못된 쿼리로 인해 생성된 결함 라인이 노출되기 시작할 수 있기 때문입니다.

느린 쿼리 로그 사용 및 CloudWatch에 게시

느린 쿼리 로그는 기본 MySQL 기능이며 성능 개선 도우미를 보완합니다. 모범 사례는 이 두 가지 방법을 모두 사용하여 인스턴스에 혼란을 야기할 수 있는 문제가 있는 쿼리를 미리 방지하는 것입니다. 느린 쿼리 로그는 동적 변수 `long_query_time`보다 느린 모든 쿼리를 로깅합니다. 클러스터 인스턴스를 다시 시작하지 않고도 이 변수를 설정할 수 있습니다.

조정 작업에서 유연성과 격리를 제공하려면 라이터 및 리더 인스턴스에 대해 별도의 파라미터 그룹을 사용합니다. 이는 읽기-쓰기 분할을 사용하는 경우 특히 중요합니다. 필요에 따라 클러스터 인스턴스에서 `long_query_time`에 대한 편안한 제한을 설정합니다. 부하를 조정하면서 임계값을 밀리초 수준으로 설정할 수 있으므로 `long_query_time` 변수에서 공격적인 값을 목표로 할 수 있습니다. 동시성이 높고 워크로드가 잘 조정되어 있으면 거의 모든 쿼리가 밀리초 내에 실행될 것입니다.

Amazon Aurora MySQL 호환 버전에서 파일에 느린 쿼리를 기록하도록 설정하면 Aurora MySQL 호환 버전이 Aurora MySQL 호환 파일 시스템에 느린 쿼리 로그를 쓰고 24시간 동안 유지합니다. 느린

쿼리 로그를 분석을 위해 더 오랜 기간 동안 유지하려면 Amazon CloudWatch에 게시합니다. 또한 CloudWatch 대시보드를 구축하여 느린 쿼리를 모니터링할 수 있습니다. 자세한 내용은 블로그 게시물 Creating an Amazon CloudWatch dashboard to monitor Amazon RDS and Amazon Aurora MySQL을 참조하세요. Amazon CloudWatch에서 느린 쿼리를 분석하는 것 외에도 Percona Toolkit의 도구인 pt-query-digest를 사용하여 느린 쿼리 로그를 프로파일링할 수 있습니다.

팀의 효율성을 높이기 위해 쿼리를 다운로드하고 프로파일링하는 이 프로세스를 자동화하도록 선택할 수도 있습니다. 팀은 자주 그리고 더 긴 간격으로 실행되는 쿼리가 있는지 확인하고 조정 우선순위를 정해야 합니다. 느린 쿼리 로그에 쿼리가 거의 기록되지 않는 상태를 목표로 하고, 워크로드를 이해하고 조정하면서 `long_query_time`을 줄여 효율을 높일 수 있습니다.

쿼리 조정 가이드

워크로드에서 문제가 있는 쿼리를 식별한 후에는 각 쿼리를 조정해야 합니다. 다음 튜닝 가이드를 사용하면 워크로드를 더 효율적으로 실행할 수 있습니다.

스캔한 행 수 최소화

기본적인 것처럼 보이지만 쿼리를 조정할 때 유용한 조언입니다. EXPLAIN 문을 사용하고 행 열을 검토하여 최적화 프로그램이 각 조인에서 스캔하는 행 수를 확인합니다. 최적의 인덱스를 만들어 스캔되는 행 수를 줄인 다음 쿼리를 다시 설명하여 제대로 작동하는지 확인합니다. 자세한 내용은 MySQL 설명서를 참조하세요.

파티션된 테이블을 사용하는 경우 최적화 프로그램이 각 파티션을 스캔하지 않아도 되도록 파티션 정리를 활성화하는 WHERE 절을 사용하여 항상 테이블을 쿼리합니다. WHERE 절에 파티션된 열의 상수가 포함되어 있는 경우 최적화 프로그램은 어떤 파티션을 찾아야 하는지 알고 있으므로 쿼리의 효율성을 높일 수 있습니다.

이 조언의 또 다른 측면은 데이터베이스 설계입니다. 쿼리에 테이블 수가 적을수록 쿼리 속도가 빨라집니다. 데이터베이스 설계를 비정규화할 수 있는 경우 최적화 프로그램이 스캔하는 행 수를 줄여 쿼리 성능을 높일 수 있습니다.

임시 테이블 사용량 및 디스크의 임시 테이블 최소화

Aurora MySQL 호환 최적화 프로그램은 인덱스에서 원하는 쿼리 결과를 직접 얻을 수 없는 경우 RAM과 디스크 모두에 임시 테이블을 생성합니다. 따라서 조정의 상당 부분은 워크로드에 적합한 올바른 인덱스를 확보하는 것입니다. 하지만 워크로드에 인덱스에만 의존할 수 없는 쿼리가 있을 수 있으므로 일부 작업은 임시 파일에서 수행될 수 있습니다. 이를 최소한으로 유지하고 디스크에 테이블이 거의 생성

되지 않는 한 괜찮습니다. 임시 테이블의 크기가 너무 커서 메모리에 저장할 수 없으면 MySQL이 디스크 테이블을 생성합니다. MySQL이 내부 임시 테이블의 크기를 확인하는 데 사용하는 로직은 두 변수 tmp_table_size와 max-heap-table-size의 값 중 더 작은 값입니다. 이러한 변수를 워크로드에 따라 최적의 값으로 조정하여 임시 테이블을 방지할 수 없는 경우 드문 경우에만 디스크로 푸시할 수 있습니다.

파일 정렬 방지

워크로드에 `ORDER BY` 쿼리가 많은 경우 이를 해결하는 가장 좋은 방법은 테이블에 올바른 인덱스를 사용하는 것입니다. 파일 정렬을 방지하려면 다중 열 인덱스가 제대로 설계되었는지 확인합니다. 이전 열을 상수로 스캔하지 않으면 해당 열에 대한 정렬이 수행되지 않습니다. `in`, `>`, `<`, `!=` 및 `BETWEEN`은 오른쪽 다음 열에 대한 정렬을 허용하지 않습니다. MySQL에서 정렬하는 최적의 방법은 쿼리에 제공된 상수 값이 포함된 열을 연속 구조의 정렬 열 왼쪽에 배치하는 다중 열 인덱스를 배치하는 것입니다. 마지막 방법으로 쿼리가 파일 정렬 없이 결과를 반환할 수 없는 경우 정렬을 애플리케이션으로 이동합니다.

동시성이 높은 집계 쿼리 실행 방지

워크로드에 애플리케이션 내 일부 기능을 충족하기 위한 소수의 집계 쿼리가 있을 수 있습니다. 이 사용 사례에는 많은 주의가 필요합니다. InnoDB 엔진은 적절한 온라인 트랜잭션 처리(OLTP) 로드에 적합하지만 높은 동시성에 대한 몇 개의 그룹별 쿼리라도 CPU에 큰 부담을 주고 클러스터의 성능을 급속히 저하시킬 수 있습니다. 집계된 결과 세트가 필요한 사용 사례를 해결하려면 쿼리별로 그룹화하는 것을 피할 수 있도록 읽을 준비가 된 테이블에 데이터를 미리 집계합니다.

쿼리의 동시성 테스트

개별 쿼리를 조정할 때 이러한 쿼리는 Aurora MySQL 호환의 여러 vCPU에서 동시에 실행된다는 점을 기억하세요. 테스트 환경에서 한 번 실행해도 쿼리가 몇 밀리초 안에 실행될 수 있습니다. 하지만 이것이 전부는 아닙니다. 프로덕션 클러스터의 예상 동시성 수준으로 쿼리를 테스트하고 성능을 벤치마크해야 합니다. 동시 실행 목표를 충족할 때만 쿼리를 프로덕션으로 릴리스하세요. 캐시에서 결과를 가져오는 것을 방지하려면 테스트 스크립트에서 최적화 프로그램 `hint sql_no_cache`를 사용해야 합니다. mysqlslap과 같은 도구를 사용하여 동시에 테스트를 수행하고 결과를 벤치마크할 수 있습니다.

쓰기 워크로드 조정

로드 밸런싱을 구현하고 라이터 인스턴스를 비우면 쓰기 작업량이 급증할 때 쓰기 워크로드 성능을 높일 수 있습니다. 동시 실행 빈도가 높을 때 쓰기 성능을 높이려면 다음 추가 단계를 따르세요.

애플리케이션 계층으로 참조 무결성 이동

참조 무결성 검사가 중요하긴 하지만 하이퍼스케일링을 사용하면 부하에 악영향을 미칠 수 있습니다. 각 쓰기에 대해 쓰기 자체를 실행하기 전에 추가 스캔을 수행해야 하므로 성능이 저하됩니다. 애플리케이션에 엄격한 무결성 검사가 필요한 경우 애플리케이션 계층에 빌드하여 쓰기 시 병목 현상이 발생하지 않도록 합니다.

무거운 프라이머리 키 사용 안 함

프라이머리 키는 가볍게 유지하세요. InnoDB 스토리지 엔진은 테이블에 생성하는 다른 모든 인덱스에 프라이머리 키를 추가합니다. 프라이머리 키가 크면 인덱스 크기에 영향을 줍니다. 프라이머리 키가 상당히 크면 데이터 페이지 저장 및 검색 속도가 느려집니다. 일반적인 예로는 범용 고유 식별자를 프라이머리 키로 사용하는 경우를 들 수 있습니다. 하이퍼스케일 환경에서 성능을 목표로 하는 경우에는 좋은 방법이 아닙니다.

파티션 교환을 사용하여 파티션된 테이블에 데이터를 로드할 수 있습니다.

파티션된 테이블에 대규모 데이터 세트를 쓰는 경우 삽입을 위해 기본 테이블에 액세스하지 않기 때문에 LOAD DATA FROM S3와 파티션 교환을 결합하면 성능이 향상될 수 있습니다. 파티션 교환에는 데이터 정의 언어(DDL)가 포함되며, 이로 인해 테이블에 메타데이터가 잠깁니다. 파티션 교환 DDL이 대기 없이 메타데이터 잠금을 확보할 수 있도록 테이블에서 실행되는 쿼리가 거의 없거나 전혀 없을 때 이 작업을 수행해야 합니다. 교환 자체를 완료하는 데 몇 밀리초 밖에 걸리지 않습니다.

사용되지 않는 인덱스 삭제

InnoDB는 데이터 증가에 따라 쿼리 계획을 최적화하므로 데이터베이스에서 사용하지 않는 인덱스가 있는지 확인하고 제거하는 것이 좋습니다. 애플리케이션이 데이터를 테이블에 쓸 때 사용하지 않는 인덱스는 IO를 소모합니다. 사용하지 않는 인덱스 목록을 확인하고 이러한 인덱스가 분기별 보고서와 같은 드문 상황에서 사용되는 인덱스가 아닌지 확인하세요. 또한 일부 인덱스은 고유성 또는 순서 지정을 적용하는 데 사용되므로 반드시 고려해야 합니다.

이전 행 버전을 효율적으로 제거해야 합니다.

MVCC(다중 버전 동시성 제어)의 InnoDB 구현에서는 레코드가 수정되면 수정 중인 데이터의 현재(이전) 버전이 먼저 실행 취소 로그에 실행 취소 레코드로 기록됩니다. 기록 목록 길이(HLL) 값이 커지면 InnoDB 가비지 수집 스레드(제거 스레드)가 쓰기 워크로드를 따라가지 못하거나 장기 실행 쿼리 또는 트랜잭션으로 인해 제거가 차단되는 것입니다. 가비지 수집이 차단되거나 지연되면 데이터베이스에서 상당한 제거 지연이 발생하여 쿼리 성능에 부정적인 영향을 미칠 수 있습니다. 다음 권장 사항을 사용하여 제거 프로세스를 최적화할 수 있습니다.

- 트랜잭션을 작게 유지하세요.
- 읽기 쿼리의 경우 READ COMMITED 격리 수준을 사용합니다.
- 제거 스레드(innodb_purge_threads 및 innodb_purge_batch_size) 수를 늘립니다. 참고로 이러한 파라미터 조정에는 재부팅이 필요합니다.
- HLL을 정기적으로 모니터링하고 가비지 수집 진행을 방해하는 워크로드 문제를 해결합니다.

로깅으로 인해 추가 경합이 발생하지 않는지 확인

일반 쿼리 로그에는 클라이언트 연결 및 연결 해제뿐만 아니라 서버에서 수신한 모든 명령문이 받은 순서대로 기록됩니다. 활성화되면 로깅이 동기식으로 이루어지므로 사용량이 많은 시스템에서 성능이 크게 저하될 수 있습니다. 필요한 경우가 아니면 일반 로그를 비활성화하는 것이 좋습니다.

느린 쿼리 로그는 long_query_time 실행 시간(기본 설정 10초)보다 오래 걸린 명령문을 기록합니다. 이 설정을 0으로 설정하면 모든 명령문이 동기적으로 로깅되므로 사용량이 많은 데이터베이스에서는 성능이 저하될 수 있습니다.

부하 제한

응답 시간을 개선하고 중요한 워크플로에 사용할 수 있는 리소스를 늘리려면 불필요한 부하를 없애야 할 수 있습니다. 이 섹션에서 다루는 대부분의 솔루션은 절충안을 위한 결정입니다. 이는 애플리케이션에 영향을 미치므로 신중하게 고려해야 합니다. 다음과 같은 경우 이러한 솔루션 사용을 고려할 수 있습니다.

- 이미 가장 큰 크기의 인스턴스를 사용하고 있습니다(특히 쓰기 가능한 기본 데이터베이스 인스턴스의 경우).
- 다른 변경 사항을 적용할 수 있는 충분한 여유 공간을 단기적으로 제공하기 위한 최후의 수단입니다.

즉각적인 변경 사항에는 다음이 포함됩니다.

- 중요하지 않은 읽기 트래픽을 프라이머리 DB 인스턴스에서 멀리 이동합니다.
- 오래된 데이터를 보관하거나 삭제합니다.
- 참조 무결성을 제거합니다.
- binlog를 비활성화합니다(사용 중인 경우).
- 중요하지 않은 추출, 전환, 적재(ETL) 프로세스를 연기합니다.
- 필수적이지 않은 애플리케이션 기능을 일시 중단하거나 저하시킵니다.

이러한 조치를 취하기 전에 장기적인 비즈니스 목표 및 위험의 맥락에서 평가하세요.

읽기 및 쓰기 워크로드 분리

MySQL 기반 애플리케이션을 실행할 때 흔히 사용되는 기법은 쓰기(프라이머리) 데이터베이스 인스턴스에서 하나 이상의 읽기 전용 데이터베이스 복제본으로 읽기 작업을 오프로드하는 것입니다. 읽기를 오프로드하면 기본 데이터베이스 인스턴스의 전체 부하를 줄이고 쓰기를 위한 공간을 확보할 수 있습니다. 복제본에 대한 즉각적인 쓰기 후 읽기 일관성에 의존하지 않는 읽기만 대상으로 해야 합니다. 보다 효율적인 방법은 모든 읽기 트래픽을 복제본으로 이동하고 복제가 지연되는 경우 쓰기 후 읽기를 재시도하도록 계획하는 것입니다. 보고 서비스와 같이 오프로드할 수 있는 독립적인 읽기 워크로드가 있습니다. 다른 읽기에는 읽기가 실행된 이유에 대한 컨텍스트가 잘 알려진 애플리케이션 수준에서 변경이 필요합니다.

또 다른 접근 방식은 애플리케이션과 데이터베이스 간의 중개자 역할을 하는 데이터베이스 프록시 솔루션을 구현하는 것입니다. 이 솔루션은 애플리케이션이 인식하지 못하는 상태에서 읽기 쓰기 분할 및 쿼리 라우팅 기능을 제공할 수 있습니다. ProxySQL, MariaDB MaxScale, ScaleARC 및 Heimdall Data는 사용 가능한 MySQL 호환 프록시 솔루션 중 일부입니다. 이러한 제품은 캐싱 또는 연결 멀티플렉싱과 같은 여러 추가 기능을 제공합니다. 트래픽이 갑자기 급증하여 애플리케이션 스택에 새로운 기술을 구현하는 경우, 의도하지 않은 부작용이 발생할 수 있는 추가 기능을 사용하기 전에 먼저 읽기/쓰기 쿼리를 분할하는 기본 기능부터 시작하여 동작과 성능을 테스트하는 것이 좋습니다.

오래된 데이터 보관 또는 삭제

데이터베이스 성능을 개선하는 또 다른 방법은 기록 데이터를 다른 테이블, 데이터베이스 또는 Amazon Simple Storage Service(S3)에 오프로드하는 방법입니다. 대부분의 데이터베이스는 애플리케이션의 전체 기록에 대한 모든 데이터를 인라인으로 보관합니다. 일반적인 사용자 대상 애플리케이션의 일반적인 상황에서 이렇게 하면 사용자가 이전 주문을 모두 볼 수 있습니다. 수요가 갑자기 급증하면 애플리케이션을 사용하는 많은 사용자가 신규 사용자이거나 신규 주문에 집중하고 있을 수 있습니다. 기록 데이터가 수십억 개의 행을 포함하는 단일 테이블에 온라인으로 존재하는 경우 상황이 더 복잡해집니다. 테이블에 큰 인덱스도 있을 수 있습니다. 테이블 데이터와 인덱스 모두 트리 구조로 저장됩니다. 테이블에 더 많은 항목을 포함하려면 트리의 수준이 더 많이 필요하므로 행에 액세스하려면 더 많은 I/O 작업이 필요합니다. 이렇게 하면 개별 레코드를 찾기 위한 액세스 시간이 늘어납니다. 더 중요한 것은 인덱스의 불필요한 부분이 데이터베이스 페이지 캐시(nnoDB 버퍼 풀)에 많이 있게 된다는 것입니다.

오래된 데이터를 별도의 테이블, 별도의 데이터베이스 또는 Amazon S3에 아카이브하면 최종 사용자의 액세스 시간을 줄이고 귀중한 캐시를 확보하고 전반적인 애플리케이션 성능을 개선할 수 있습니다. 이벤트 전에 오래된 데이터를 보관(예: 30일 또는 90일만 유지)하면 중요한 테이블의 테이블 및 인덱스

크기가 제한됩니다. 이 변경 사항을 적용하려면 기록 데이터를 보도록 명시적으로 요청하는 소수의 사용자에 대해서만 보조 위치에서 이전 데이터를 쿼리하도록 애플리케이션을 수정해야 합니다.

중요하지 않은 ETL 프로세스 연기

기본 데이터베이스 시스템에서 ETL 프로세스를 위해 데이터를 추출하면 대규모 환경에서 트랜잭션 및 동시 작업이 많이 발생하는 경우 안정성 위험이 발생할 수 있습니다. ETL 프로세스는 다음과 같은 특징으로 알려져 있습니다.

- 장기 실행 트랜잭션
- 광범위한 잠금
- CPU, 메모리 및 I/O 소비
- 중요한 최종 사용자 경로를 제공하는 트랜잭션 워크로드와의 경합.

가변적이거나 예측할 수 없는 ETL 프로세스(예: 쿼리 `INSERT INTO ... SELECT FROM ...;`)는 부하 및 경합의 전반적인 변동성을 가중시켜 안정성 위험을 증가시킵니다.

가능하면 ETL 프로세스 실행 빈도를 연기하거나 줄입니다. 특히 중요한 기능을 제공하지 않는 경우에는 더욱 그렇습니다. 중요한 ETL 프로세스의 경우, 고정된 수의 행(예: `LIMIT` 절 사용)의 처리 배치와 같은 제한된 작업 단위에서 작동하도록 수정하거나, 별도의 트랜잭션을 사용하거나, DB 인스턴스의 최대 리소스 요구량을 줄이기 위해 장기간에 걸쳐 더 적은 양의 데이터를 끌어옵니다.

덜 중요한 애플리케이션 기능 끄기

급증하는 요청을 처리할 때 사용자 환경을 적절하게 저하시키거나 비핵심 기능을 제거하면 중요한 기능을 위한 데이터베이스 리소스를 절약할 수 있습니다. 이렇게 하려면 고객 환경에 약간의 변화가 필요할 수 있지만 사이트가 다운되는 것보다는 다른 흐름을 사용하는 것이 좋습니다. 항상 데이터베이스 호출을 수행하는 사이트 첫 페이지의 개인 설정 기능을 일시적으로 비활성화할 수도 있습니다. 또는 제품을 선택할 때 재방문 고객에게 맞춤형 오퍼를 제시하는 것을 중단할 수 있습니다. 기능을 일시적으로 끄면 데이터베이스에 액세스할 필요 없이 페이지를 캐시하거나 전달할 수 있습니다.

다른 예로는 데이터베이스 호출이 필요한 데이터 변경 사항을 확인하는 폴링 메커니즘이 있는 프런트엔드가 있습니다. 폴링 빈도를 줄이면 즉시 데이터베이스 호출 수가 감소합니다. 페이지 매김이 필요하거나 상위 결과에서 멀리 떨어진 정렬된 결과 세트를 검색할 수 있는 옵션을 사용자에게 제공하는 사용자 인터페이스의 경우 데이터베이스 호출 비용이 점점 더 증가합니다. 가장 비용이 많이 드는 데이터베이스 호출로부터 데이터베이스를 보호하려면 결과 세트의 페이지 수를 제한합니다. 애플리케이션 계

층의 기능 플래그를 사용하면 애플리케이션 또는 데이터베이스 부하가 증가할 때 운영 팀이 기능을 끄거나 성능을 저하시킬 수 있습니다.

파라미터 튜닝

연결 관련 파라미터 외에도 과성장에 직면했을 때 Amazon Aurora MySQL 호환 버전 클러스터의 성능을 개선하기 위해 조정할 수 있는 몇 가지 파라미터가 있습니다. 데이터베이스 파라미터를 변경할 때는 다음 모범 사례를 사용하는 것이 중요합니다.

- 변경의 영향을 측정할 수 있도록 한 번에 하나의 파라미터를 변경합니다.
- 일부 파라미터는 변경된 후 효과가 나타나기 위해 워밍업 기간이 필요할 수 있습니다. Aurora MySQL 호환 클러스터의 성능을 관찰하고 측정할 때는 이 점을 고려하세요.
- 데이터베이스 인스턴스 시작에 방해가 될 수 있는 잘못된 파라미터 값을 사용하지 마세요.

파라미터는 다음과 같습니다.

- `sync_binlog`
- `table_open_cache`
- `innodb_sync_array_size`
- `innodb_flush_log_at_trx_commit`
- `autocommit`
- `tmp_table_size`
- `max_heap_table_size`

이러한 파라미터 구성에 대한 지침은 AWS 설명서의 Aurora MySQL 구성 파라미터, Aurora MySQL의 MySQL 기능에 대한 권장 사항 및 Aurora MySQL의 임시 테이블 동작을 참조하세요.

리소스

- Journey to Adopt Cloud-Native Architecture Series: #1 – Preparing your Applications for Hypergrowth(블로그 게시물)
- Journey to Adopt Cloud-Native Architecture Series: #2 – Maximizing System Throughput(블로그 게시물)
- Amazon RDS 프록시 사용
- 캐싱 전략
- 성능 개선 도우미 설정 및 해제
- InnoDB 스토리지 엔진
- Aurora 복제본
- MariaDB 커넥터/J
- MariaDB MaxScale
- ProxySQL
- Heimdall Data

문서 기록

아래 표에 이 가이드의 주요 변경 사항이 설명되어 있습니다. 향후 업데이트에 대한 알림을 받으려면 RSS 피드를 구독하십시오.

변경 사항	설명	날짜
최초 게시	—	2022년 10월 5일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.