Managing Redis - Optimization

생성일	@2023년 1월 7일 오후 6:34
태그	

Redis benchmark

redis-benchmark 유틸리티를 이용해 M개의 총 쿼리를 보내는 동시에 N개의 클라이언트가 수행하는 것을 테스트 할 수 있다. 이 유틸리티는 기본 데이터 세트를 제공하거나 사용자 지정 테스트 세트를 제공할 수 있다.

Selecting the size of the key space

기본적으로 벤치마크는 단일 키에 대해 실행된다. 하지만 복합적인 키를 사용해서 벤치마크를 할 수 있는 방법이 있다.

Using pipelining

파이프라이닝을 사용하는 경우도 벤치마크 테스트 가능하다.

Pitfalls and misconceptions

동일한 워크로드와 동일한 버전에서 비교할때 유효하다. 다른 옵션으로 비교할 계획이면 기능 및 기술 차이점을 평가하고 고려하는 것이 중요하다.

- Redis는 서버이기 때문에 임베디드 저장소와 비교하는 것은 의미가 없다.
- Redis명령은 모든 일반 명령에 대한 승인을 반환한다. 다른 데이터 저장소는 그렇지 않다. 단방향 쿼리와 관련된 저장소와 비교하는 것은 조금만 유용하다.
- 실제로 테스트하려면 redis-benchmark를 이용하던가 파이프라이닝을 이용해 여러 명령 및 여러 스레드/프로세스를 집계해야한다.
- 트랜잭션 서버(MySQL, PostgreSQL 등)와 비교하려면 AOF를 활성화하고 적합한 fsync 정책을 결정해야한다.
- CPU코어수만큼 Redis 인스턴스를 실행해야지 공평하다.

redis-benchmark의 기본동작은 동시성만 활용해서 처리량을 달성하는 것이다. 실제로 병렬처리나 파이프라이닝을 할거면 옵션을 줘야한다.

Factors impacting Redis performance

Redis 성능에 직접적인 영향일 미치는 요소가 있다.

- 네트워크 대역폭과 대기 시간
- CPU
- RAM의 스피드와 대역폭
- VM에서 실행될 때 더 느리다.
- TCP/IP 루프백과 유닉스 도메인 소켓을 모두 사용 가능하다. 약 50% 더 많은 처리 가능하다.
- Unix 도메인 소켓은 파이프라인이 많이 사용되는 경우 성능이점이 감소하는 경향이 있다.

Other things to consider

또 다른 고려사항들이 있다.

- 가능한 격리된 하드웨어에서 테스트를 실행하는 것이 좋다.
- 가변 CPU 코어 주파수 매커니즘이 있는 경우 가장 높은 고정 주파수를 사용하는 것이 좋다.
- 벤치마크에 따라 시스템의 size를 조정해야한다. 충분한 RAM, 스왑 가능 설정.
- RDB 또는 AOF를 테스트 할때 시스템이 다른 I/O활동이 없는지 체크해야한다.
- Redis 로깅 수준을 경고 또는 알림으로 설정하고 생성된 로그 파일을 원격 파일 시스템에 두면 안된다.
- 벤치마크 결과를 변경할 수 있는 모니터링 도구를 사용하면 안된다. INFO는 되지만 MONITOR는 안된다.

Redis CPU profiling

Filling the performance checklist

Ensuring the CPU is your bottleneck

시스템의 상태를 완전히 확인하고 CPU 병목을 식별했다고 가정한다.

Build Prerequisites

On-CPU 분석을 위해 Redis는 추적 프로그램에서 추적 가능하도록 수정해야한다.

기본적으로 Redis는 컴파일러 최적화가 활성화 된채로 컴파일 되는데, 이는 실행속도는 빨라지지만 추적하기가 어려워지는 단점이 있다. 적절한 옵션을 줘서 테스트 용이하게 만들어 보자

A set of instruments to identify performance regressions and/or potential on-CPU performance improvements

on-CPU 리소스 병목 현상 분석에 중점을 둔다. 이를 위해 툴킷 및 하드웨어별 PMC(성능 모니터링 카운터)를 사용해 진행한다.

- 핫스팟 분석
- 호출횟수 분석
- 하드웨어 이벤트 샘플링

Diagnosing latency issues

> 느린 응답의 이유찾기

여기서의 응답은 클라이언트가 명령을 실행하는 시간과 명령에 대한 응답을 받는 시간 사이의 최대 지연이다. 일반적으로 Redis 처리 시간은 매우 빠르지만 대기시간이 더 높아지는 특정 조건이 있다.

I've little time, give me the checklist

짧은 대기 시간 방식으로 Redis를 실행하는 방법을 위한 체크리스트

- 서버를 차단하는 느린 명령을 실행하지 않는지 확인. 확인하려면 Redis Slow Log 기능 사용
- EC2 사용자의 경우 HVM기반 최신 인스턴스를 사용해야한다. 안그러면 fork()가 느리다
- 커널에서 transparent huge page를 비활성화해야한다.
- 가상 머신을 사용하는 경우 Redis와 관련이 없는 본질적인 대기 시간이 있을 수 있다.
- redis 서버에서 ./redis-cli —intrinsic-latency 100을 사용해 런타임 환경에서 기대할 수 있는 최소 지연 시간을 확인해라.

- 대기시간 모니터링 기능을 활성화하고 이를 통해 대기시간 이벤트 및 원인에 대해 파악해라

내구성 대 대기시간/성능 트레이드 오프에 대해 다음을 참조

- AOF + 항상 fsync: 매우느림. 수행중인 작업을 알고있는 경우에만 사용
- AOF + 매초 fsync: 제일 좋은 절충안
- AOF + 매초 fsync + no-appendfsync-on-rewrite yes옵션: 매우 좋은 절충안이지만 디스크 부하를 낮추기 위해 재작성 중에 fsync 안함
- AOF + fsync사용안함: fsyncing은 이 설정에서 커널에 달려있고, 디스크 부하와 대기시간 스파이크 위험이 훨씬 적다.
- RDB: 구성하는 저장 트리거에 따라 다양한 장단점이 있다.

Redis latency monitoring

> Redis에서 느린 서버 이벤트 발견

Redis는 단일스레드 이므로 일반적으로 코어당 수행할 수 있는 작업량과 제공할 수 있는 대기 시간 수치의 관점에서 이점이다. 그러나 단일 스레드가 제공되는 다른 클라이언트에 영향을 미치지 않는 방식으로 특정 작업을 점진적으로 수행할 수 있어야 하므로 대기시간 문제가 발생한다.

레이턴시 모니터링으로 대기시간 문제를 확인하고 해결하는데 도움이 된다.

- 다양한 레이턴시에 민감한 코드 경로를 샘플링하는 대기시간 후크
- 여러 이벤트로 분할된 대기시간 스파이크의 기록.
- 시계열에서 원시 데이터를 가져오는 보고 엔진
- 측정에 따라 사람이 읽을 수 있는 보고서와 힌트를 제공하는 분석 엔진

Events and time series

다른 모니터링 코드 경로는 다른 이름을 가지며 이를 이벤트라고 말한다.

- command: 느린 명령 실행의 대기 시간 스파이크 측정 이벤트
- fast-command: O(1) 및 O(logN)명령 모니터링을 위한 이벤트
- 등등

대기 시간 스파이크는 구성된 대기 시간 임계값보다 실행하는 데 더 많은 시간이 걸리는 이벤트이다. 모니터링되는 모든 이벤트와 연결된 별도의 시계열이 있다. 시계열이 작동하는 방식은 다음과 같다.

- 대기시간 스파이크가 발생할 때마다 적절한 시계열에 기록
- 모든 시계열은 160개의 요소로 구성
- 각 요소는 대기시간 스파이크가 측정된 시간의 Unix 타임스탬프와 이벤트 실행에 걸린 시간으로 구성된 쌍이다.
- 같은 초에 발생하는 동일한 이벤트에 대한 지연 스파이크는 최대 지연 시간을 선택해서 병합된다. 낮은 임계값에서 발생할 수 잇는 특정 이벤트에 대해 연속 대기시간 스파이크가 측정되더라도 최소 160초의 기록을 사용할 수 있따.
- 모든 요소에 대한 상시 최대 대기 시간을 기록한다.

How to enable latency monitoring

대기시간 임계치를 밀리초 단위로 설정하면 해당 임계치를 넘는 명령은 기록된다. 모니터링을 시작하는데 드는 하드웨어적 비용은 0에 가까워서 추가로 RAM을 할당할 필요는 없다.

Memory optimization

> 레디스에서 메모리 최적화 유형의 전략들

Special encoding of small aggregate data types

많은 데이터 유형이 특정 크기까지 더 적은 공간을 사용하도록 최적화 되었다. (해시, list, 정수로 구성된 sets) 이는 CPU/Memory 트레이드 오프이기 때문에 특정크기를 redis.conf의 지시문을 이용해 변경할 수 있다.

특별히 인코딩 된 값이 구성된 최대 크기를 초과하면 Redis가 자동으로 일반 인코딩으로 변환한다.

Using 32 bit instances

32bit 로 컴파일된 Redis는 포인터가 작아서 메모리를 적게 사용하지만 최대 메모리 사용량이 4GB로 제한된다. RDB와 AOF파일은 32bit와 64bit 인스턴스 간에 호환되므로 문제없이 전환 가능하다.

Bit and byte level operations

비트와 바이트 수준 작업 명령이 도입되었다. 이 명령을 사용해 Redis 문자열 유형을 임의 엑세스 배열로 처리할 수 있다. 비트맵 연산같은 작업 할 수 있다.

Use hashes when possible

작은 해시는 매우 작은 공간에 인코딩되므로 가능한 한 해시를 사용해 데이터를 표현해야한다.

Using hashes to abstract a very memory efficient plain key-value store on top of Redis

몇 개의 키는 몇 개의 필드가 있는 해시를 포함하는 단일 키보다 훨씬 더 많은 메모리를 사용한다. 이론적으로는 우리가 조회를 일정한 시간에 수행하도록 보장하기 위해 평균적인 경우에는 해시 테이블과 같이 일정한 시간 복잡도를 갖는 데이터 구조를 사용해야한다.

Memory allocation

사용자 키를 저장하기 위해 Redis는 maxmemory 설정이 활성화하는 만큼의 메모리를 할당한다. 정확한 값은 구성 파일이나 CONFIG SET명령으로 설정할 수 있다. Redis가 메모리를 관리하는 방법에 대해 주목해야할 몇가지 사항이 있다.

- 키가 제거될 때 OS에 메모리를 항상 해제하지는 않는다.
- 피크 메모리 사용량에 따라 메모리를 프로비저닝해야한다. 워크로드에 10GB가 필요한 경우 대부분이 5GB로 가능하더라도 10GB를 프로비저닝해야한다.
- 하지만 5GB중에 2GB의 키를 해제하고 2GB 만큼의 추가 키를 추가하면 재사용 가능하다.
- 조각화는 실제로 사용된 물리적 메모리(RSS)를 현재 사용중인 메모리 양으로 나눈 값으로 계산된다.

maxmemory가 설정되지 않은 경우 모든 여유 메모리를 소모할 수 있다. 그래서 일부 제한을 구성하는 것이 좋다. 한계에 다다르면 메모리 부족 오류를 반환하지만 전체 시스템이 죽지는 않는다.