

005. Undertow Io Thread, Worker Thread와 Hikari CP

XNIO workers

All listeners are tied to an XNIO Worker instance. Usually there will only be a single worker instance that is shared between listeners, however it is possible to create a new worker for each listener.

The worker instance manages the listeners IO threads, and also the default blocking task thread pool. There are several main XNIO worker options that affect listener behaviour. These options can either be specified on the Undertow builder as worker options, or at worker creating time if you are bootstrapping a server manually. These options all reside on the `org.xnio.Options` class.

WORKER_IO_THREADS

The number of IO threads to create. IO threads perform non blocking tasks, and should never perform blocking operations because they are responsible for multiple connections, so while the operation is blocking other connections will essentially hang. Two IO threads per CPU core is a reasonable default.

WORKER_TASK_CORE_THREADS

The number of threads in the workers blocking task thread pool. When performing blocking operations such as Servlet requests threads from this pool will be used. In general it is hard to give a reasonable default for this, as it depends on the server workload. Generally this should be reasonably high, around 10 per CPU core.

IO Thread

Non-blocking 작업을 수행하는 주체이며 절대 blocking 되는 작업을 수행해선 안된다.

IO Thread의 수는 Core size * 2 정도가 합리적이다

Worker Thread

Servlet 환경 등에서 요청에 대한 처리를 담당하는 주체로 blocking 작업이 worker에 의해 수행 된다.

서버의 역할에 따라 달라지기에 절대적인, 합리적인 기본값은 제공되지 않는다.

Worker Thread의 수는 Core * 10 정도로 상당히 높아야 한다.

Undertow에서 Io thread와 Worker thread 수를 조정하는 로직

```

private Builder() {
    ioThreads = Math.max(Runtime.getRuntime().availableProcessors(), 2);
    workerThreads = ioThreads * 8;
    long maxMemory = Runtime.getRuntime().maxMemory();
    //smaller than 64mb of ram we use 512b buffers
    if (maxMemory < 64 * 1024 * 1024) {
        //use 512b buffers
        directBuffers = false;
        bufferSize = 512;
    } else if (maxMemory < 128 * 1024 * 1024) {
        //use 1k buffers
        directBuffers = true;
        bufferSize = 1024;
    } else {
        //use 16k buffers for best performance
        //as 16k is generally the max amount of data that can be sent in a single write() call
        directBuffers = true;
        bufferSize = 1024 * 16 - 20; //the 20 is to allow some space for protocol headers, see UNDERTOW-1209
    }
}
}

```

→ io thread: core size (min 2)

→ worker thread: io thread * 8

기본적으로 undertow에선 코어 사이즈만큼 IO thread를 가져가고 worker thread는 io thread의 8배를 가져간다.

따라서 상황에 따라 커스터마이징이 필요하다.

SITN-API, Accounts-API의 ECS fargate에서 설정하는 Memory, CPU는 모두 아래와 같다

작업 메모리(MiB)	2048
작업 CPU(단위)	1024

즉 core는 1개이다.

따라서 최적화 된 설정은 아래와 같다.

Io thread: 2개

worker thread: 10개 ~ 16개

그렇기에 각 application에서 최적화를 해 줘야 할 것 같지만 실제로 아래 구문의 호출 결과는 core * 2가 나온다.

`Runtime.getRuntime().availableProcessors()`

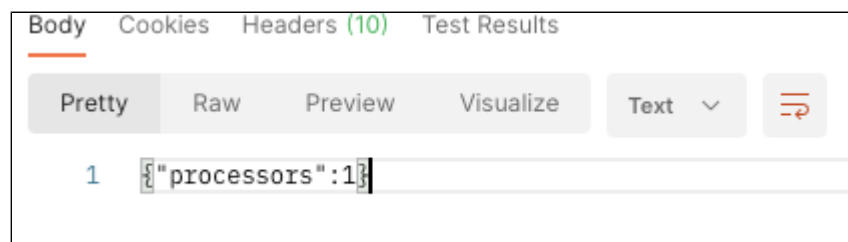
인텔 CPU의 하이퍼스레딩이라는 것을 통해 물리적 코어 한개당 두개의 논리적 프로세서를 할당 해주었기 때문이다.

개발 Mac은 아래와 같이 설정 된다.



Fargate의 CPU는 하이퍼스레딩이 지원 될까?

아쉽게도 지원하지는 않는듯 하다.



하지만 최소 io thread size는 2개니 크게 문제는 없을 것 같다.

Hikari CP의 pool size는 어떻게 조정 할까? (<https://github.com/brettwooldridge/HikariCP/wiki/About-Pool-Sizing#pool-locking>)

보통 worker thread의 개수로 비율을 잡아 산정하는데 Hikari 공식문서에선 아래와 같은 공식을 소개한다.

공식에 따르면 $\text{core count} * 2 + \text{effective_spindle_count}$ (16개 DISK가 있는 RAID= 16, 한 번에 처리 가능한 I/O 수 OR HDD 개수를 의미하는 듯)이다.

여기서 core_count는 하이퍼스레딩을 지원하는 CPU인 경우도 무시해야 한다.

따라서 $1 * 2 + ??$ 일텐데 개수가 너무 작다

The Formula

The formula below is provided by the PostgreSQL project as a starting point, but we believe it will be largely applicable across databases. You should test your application, i.e. simulate expected load, and try different pool settings *around* this starting point:

$\text{connections} = ((\text{core_count} * 2) + \text{effective_spindle_count})$

그래서 deadlock을 피할 수 있는 최소 개수 공식에 따르면 $\text{thread} * (\text{thread per connection} - 1) + 1$ 이다.

$$\text{pool size} = T_n \times (C_m - 1) + 1$$

Where T_n is the maximum number of threads, and C_m is the maximum number of *simultaneous connections* held by a single thread.

하나의 요청에서 요구하는 커넥션 수가 API 마다 다르지만 Study 생성 API를 기준으로 보면 8개의 쿼리가 나간다. Create API의 가중치를 1로 두고

Study Retrieve API는 1개의 쿼리가 나간다. 가중치를 3으로 두면 총 13

$$16 * (13/4 - 1) + 1$$

$$\rightarrow 16 * (3.25 - 1) + 1$$

$$\rightarrow 16 * 2.25 + 1 = 37$$

올림하여 40개정도가 적당한 것 같다.

그 외 Connection Pool 옵션

프 로 퍼 티	설명

connectionTimeout	<p>Connection pool에 커넥션을 요청 했을 때 최대 얼마나 기다릴 지에 대한 대기시간을 의미한다.</p> <p>deadlock 등으로 지정된 시간에 커넥션을 얻어오지 못 한 경우 Exception이 떨어진다.</p>
validationTimeout	<p>커넥션이 유효한지 검사할 때 대기 시간을 지정한다. 이 값은 connectionTimeout보다 작아야 한다. 허용 가능한 최소 값은 250이다.</p>
idleTimeout	<p>커넥션이 풀에서 유휴 상태로 남을 수 있는 최대 시간을 밀리초 단위로 지정한다. 이 값은 minimumIdle이 maximumPoolSize보다 작아야 한다. 이 값이 지정된 경우 요청이 몰려 Connection이 늘어났을 때 사용하지 않고 일정시간 idle 상태가 된 Connection들을 다시 그 때에 일정시간을 계산하는 데에 적용 하는 값이다.</p> <p>이 타임아웃 전에는 Idle Connection을 제거하지 않는다.이 값이 0이면 Idle Connection을 풀에서 제거하지 않는다.</p> <p>허용 가능한 최소 값은 10000(10초)이다.</p>

maxLifetime

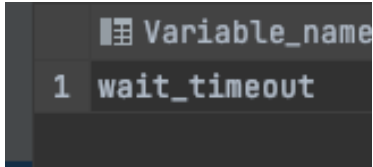
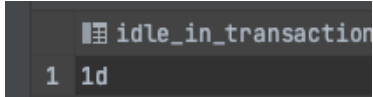
Connection이 Pool에서 유지 될 최대 시간을 의미한다. 이 시간이 지난 커넥션 중에서 사용중인 커넥션은 종료된 이후에 풀에서 제거
갑자기 풀에서 많은 커넥션이 제거되는 것을 피하기 위해 negative attenuation(감쇠)를 적용해 점진적으로 제거한다.
houseKeepingExecutorsService에 의해 MaxLifetime의 2.5% 시간에 한번 씩 스케줄이 돈다.

```
if (maxLifetime > 0) {
    // variance up to 2.5% of the maxlifetime
    final long variance = maxLifetime > 10_000 ? ThreadLocalRandom.current().nextLong( max
    final long lifetime = maxLifetime - variance;
    poolEntry.setFutureEol(houseKeepingExecutorService.schedule(
        () -> {
            if (softEvictConnection(poolEntry, "(connection has passed maxLifetime)", false
                addBagItem(connectionBag.getWaitingThreadCount());
            }
        },
        lifetime, MILLISECONDS));
}
```

이 값을 설정할 것을 권장한다. DB나 네트워크 환경 등에서 설정 한 Connection 제한 시간 보다 최소한 5초는 짧아야 한다..

이 값이 0이면 풀에서 제거하지 않지만 idleTimeout은 적용된다.

DBMS 마다 Connection이 유지되는 시간에 대한 값이 매우 다양한 이름을 가진다..

DBMS	프로퍼티	
MariaDB	show global variables LIKE 'wait_timeout' https://mariadb.com/docs/reference/mdb/system-variables/wait_timeout/#skysql	 단위: s 기본값이 엔진 버전마다 다름 (aws: 2)
PostgreSQL	show idle_in_transaction_session_timeout	 <input type="checkbox"/> idle_in_transaction_session_tir 단위: ms 기본값이 엔진 버전마다 다름 (aws: 8)

max mu mP ool Size	<p>유휴 상태(idle)와 사용중인 커넥션을 포함해서 Connection Pool이 허용하는 최대 커넥션 개수를 설정한다</p> <p>Pool의 모든 Connection이 사용중인 경우 (idle 상태의 Connection이 없는 경우) connectionTimeout이 지날 때까지 getConne</p>
mi ni mu ml dle	<p>Connection Pool에 유지할 커넥션의 최소 개수를 설정한다.</p>
co nn ect ion Te st Qu ery	<p>Connection이 유효한지 검사할 때 사용할 쿼리를 지정한다.</p> <p>JDBC4를 지원하는 드라이버이면 이 프로퍼티를 설정하지 않는것이 좋다.</p> <p>JDBC4는 Conneciton.isValid()를 사용해서 유효한지 검사를 수행한다.</p>

표준이 잘 정의되지 않은 환경이라면 웬만해선 기본값을 그대로 쓰는것이 골치가 덜 아플 듯 하다...