venkateshcm.com

How To Determine Web Application Thread Pool Size

Venkatesh CM

6-7 minutes

Continuing on Architectural Issues faced while scaling web applications, in this blog I will cover a common issue, how to determine web application thread pool size?, that shows up while deploying web applications to production or while performance testing web applications.

Thread Pool

In web applications thread pool size determines the number of concurrent requests that can be handled at any given time. If a web application gets more requests than thread pool size, excess requests are either queued or rejected.

Please note concurrent is not same as parallel. Concurrent requests are number of requests being processed while only few of them could be running on CPUs at any point of time. Parallel requests are number of requests being processed while all of them are running on CPUs at any point of time.

In Non-blocking IO applications such as NodeJS, a single thread (process) can handles multiple requests concurrently. In multi-core CPUs boxes, parallel requests can be handled by increasing

number of threads or processes.

In blocking IO applications such as Java SpringMVC, a single thread can handle only one request concurrently. To handle more than one request concurrently we have to increase the number of threads.

CPU Bound Applications

In CPU bound applications thread Pool size should be equal number of cpus on the box. Adding more threads would interrupt request processing due to thread context switching and also increases response time.

Non-blocking IO applications will be CPU bound as there are no thread wait time while requests get processed.

IO Bound Applications

Determining thread pool size of IO bound application is lot more complicated and depends on response time of down stream systems, since a thread is blocked until other system responds. We would have to increase the number of threads to better utilise CPU as discussed in Reactor Pattern Part 1 : Applications with Blocking I/O.

Little's law

Little's law was used in non technology fields like banks to figure out number of bank teller counters required to handle incoming bank customers.

Little's law

```
The long-term average number of customers in a stable
system L is equal to the long-term
```

```
average effective arrival rate, λ, multiplied by the
average time a customer spends
in the system, W; or expressed algebraically:
L = λW.
```

Little's law applied to web applications

```
The average number of threads in a system (Threads)
is equal average web request
arrival rate (WebRequests per sec), multiplied by the
average response time (ResponseTime)
```

Threads = Number of Threads
WebRequests per sec = Number of Web Requests that can be processed in one second
ResponseTime = Time taken to process one web request

```
Threads = (WebRequests/sec) X ResponseTime
```

While the above equation provides the number of threads required to handle incoming requests, it does not provide information on the threads to cpu ratio i.e. how many threads should be allocated on a given box with x CPUs.

Testing to determining Thread Pool size

To find right thread pool size is to balance between throughput and response time. Starting with minimum a thread per cpu (Threads Pool Size = No of CPUs) , application thread pool size is directly proportional to the average response time of down stream systems until CPU usage is maxed out or response time is not degraded.

Below diagrams illustrate how number of requests, CPU and Response Time metrics are connected.

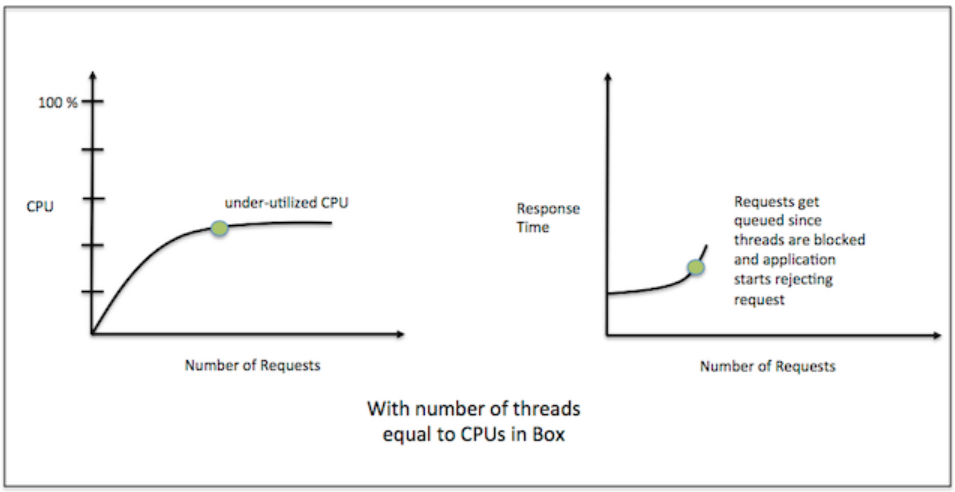
CPU Vs Number of Requests graph shows how CPU usage while

increasing load on the web applications.

Response Time Vs Number of Requests graph shows response time impact due to increasing load on the web applications.

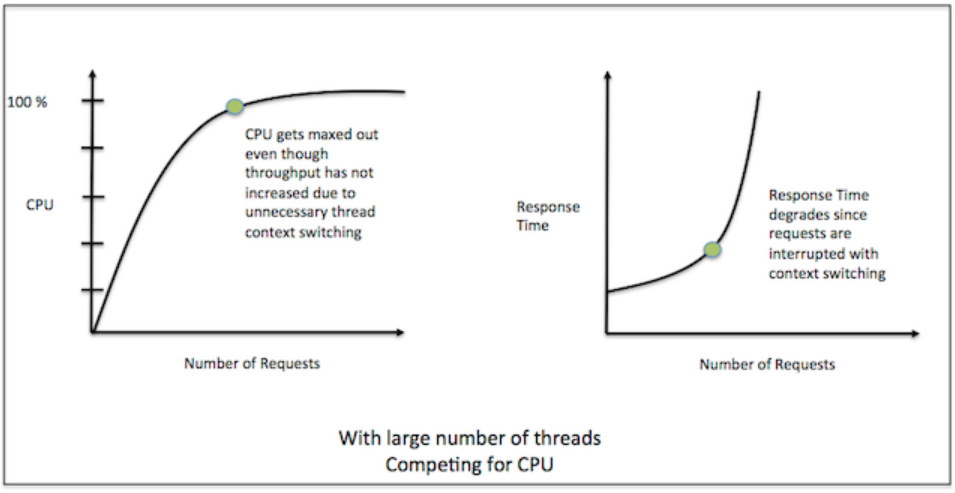
Green dot indicates point of optimal throughput and response time.

Thread pool size = Number of CPUs



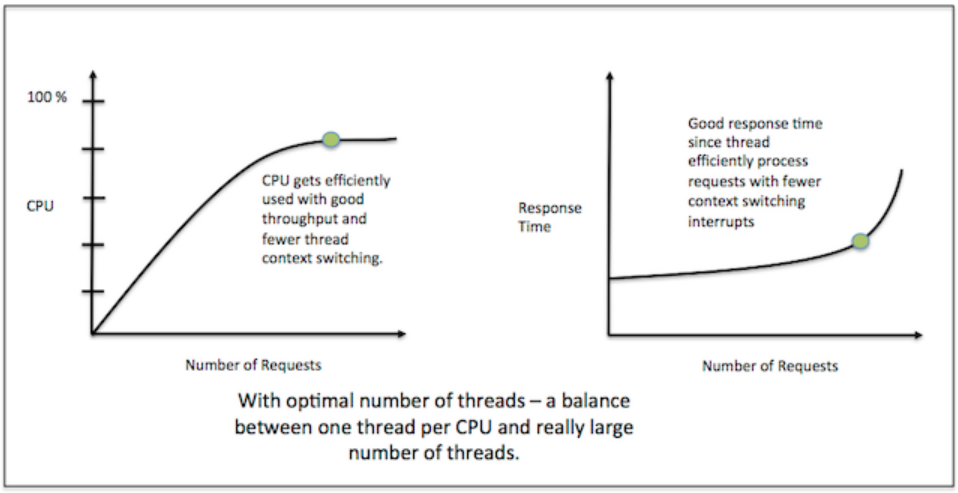
Above diagram depicts blocking IO bound applications when number of threads is equal to number cpus. Application threads get blocked waiting for down stream systems to respond. Response time increases as requests get queued since threads are blocked. Application starts rejecting requests as all threads are blocked even though CPU usage is very low.

Large Thread pool size



Above diagram depicts blocking IO bound applications when large number of threads are created in web application. Due to large number of threads, thread context switching will be very frequent. Application CPU usage gets maxed out even though throughput has not increased due to unnecessary thread context switching. Response Time degrades since requests are interrupted with context switching.

Optimal Thread pool size



Above diagram depicts blocking IO bound applications when optimal number of threads are created in web application. CPU gets efficiently used with good throughput and fewer thread context switching. We notice good response time due to efficient request processing with fewer interrupts (context switching).

Thread Pool isolation

In most web applications, few types of web request take much longer to process than other web request types.The slower web requests might hog all threads and bring down entire application.

Couple of ways to handle this issue is

- to have separate box to handle slow web requests.

- to allocate a separate thread pool for slow web requests within the same application.

  Determining optimal thread pool size of a blocking IO web application is difficult task. Usually done by performing several performance runs. Having several thread pools in a web application further complicates the process of determining optimal thread pool size.