Taller 2

Usuarios concurrentes en un servicio

Sebastian Galindo

Ing. César Julio Bustacara Medina
Arquitectura de Software

Facultad de ingeniería
Pontificia Universidad Javeriana
Colombia
Agosto 2024

Contents

Chapter 1

Introducción

Para ciertos sistemas es importante la cantidad de usuarios concurrentes que puede llegar a tener activos sin que el sistema se degrade de cierto modo. Con base en esto, desde la arquitectura se tiene que saber si es necesario desplegar un cluster de servidores o un solo servidor es suficiente para atender la cantidad de solicitudes concurrentes deseadas.

En este taller, se va a evaluar cuantos usuarios concurrentes pueden hacer una solicitud a un servicio sin que este deje de esta operativo o se degrade.

Chapter 2

Metodología

Para el desarrollo del taller se va a utilizar un computador con las siguientes características:

- **CPU**: AMD Ryzen 5 5600X 6-Core Processor 12 Threads
- **RAM**: 16258MB ≈ 16GB
- **OS**: Ubuntu 22.04.4 LTS

Para efectos de comparación, se van a realizar un total de 4 implementaciones (3 en el lenguaje de JavaScript y 1 en el lenguaje de C++). Cada una de las implementaciones toma un caso posible para recibir peticiones de manera concurrente, las implementaciones están marcadas con un nombre unico que hace referencia a la variación de la implementación.

- taller_2_threadsJS.
- taller_2_workerpool.
- taller_2_cluster.
- taller_2_cpp.

Para cada una de las implementaciones se va a realizar un benchmarking para simular las conexiones de manera concurrente para consumir el servicio. Se usaron las siguientes herramientas con sus configuraciones:

- **Apache HTTP server benchmarking tool**.
  - Para esta herramienta se van a tomar los parámetros de 20000 solicitudes para un máximo de 20000 conexiones concurrentes al servidor HTTP, estas configuraciones son el limite superior de la herramienta. Esta configuración determina que para cada una de las conexiones, se va mandar una solicitud.

- **Bombardier**
  - Esta herramienta no tiene un limite exacto para las conexiones concurrentes al servidor, por lo tanto, tanto las solicitudes como las conexiones concurrentes van a depender de la implementación y su capacidad de procesamiento.
  - Bombardier provee una opción para mandar solicitudes indefinidamente en una intervalo de tiempo con conexiones concurrentes.

Cada una de las herramientas posibilita la simulación de una conexión con una solicitud al servidor HTTP objetivo. Cada una de las implementaciones tiene un único endpoint `/asyncRequest` el cual retorna un texto plano en la respuesta HTTP.

2.1 Implementación `taller_2_threadsJS`

Esta implementación consiste en la generación y manejo de hilos *manualmente* o sin el uso de alguna librería o técnica. Cuando llega una petición al endpoint, se crea un nuevo hilo y procesa la respuesta; cuando la respuesta esta lista, el hilo envía un mensaje al hilo principal con el resultado de la petición y el hilo principal se encarga de formar la respuesta HTTP.

2.2 Implementación `taller_2_workerpool`

Esta implementación utiliza una librería especializada para el manejo de hilos. La librería **Workerpool** implementa el patrón **Thread pool** o también llamado **replicated workers / worker-crew model**[1]. Este patrón crea una serie de hilos en el sistema los cuales están esperando a que una tarea se les sea asignada mediante una cola de tareas. Cuando llega una petición HTTP, se envía la tarea a la cola para que algún hilo capture la tarea y procese la solicitud, cuando la petición esta lista, esta posteriormente se retorna al hilo principal para que construya la respuesta HTTP.

2.3 Implementación `taller_2_cluster`

La implementación de un *cluster* es una técnica usada en el entorno de NodeJS para poder aprovechar al máximo los procesadores/hilos disponibles del CPU. Esta técnica consiste en crear un proceso por cada hilo/procesador activo del CPU y a cada uno de los procesos se les captura los mensajes que envían, sin embargo, cada proceso es capaz de responder a las peticiones entrantes. Los procesos comparten el puerto del servidor y el proceso principal del cluster utiliza Round Robin para poder distribuir la carga a lo largo del cluster.

2.4 Implementación taller_2_cpp

Este modulo del taller usa una implementación externa (Codigo Fuente) de un servidor web con soporte para la ejecución en varios hilos. Según la documentación del servidor: *A total of 12 threads are running. A threadpool of initial size (10) + one scheduler thread + main thread.* El servidor usa la técnica de `First come first served` para asginar la carga a cada uno de los hilos disponibles del sistema, este hilo se debe encargar de mandar la respuesta por el socket del cliente y cerrar la conexión.

Chapter 3

Resultados

Para cada una de las implementaciones se va a ejecutar las dos herramientas descritas en la metodología, se van a capturar alguna información importante como las solicitudes por segundo o el numero de conexiones cerradas por error.

3.1 Implementación `taller_2_threadsJS`

Para el caso de esta implementación, no se puede probar con el máximo rango de las herramientas, ya que aproximadamente con las 1500 conexiones concurrentes se comienza a degradar el sistema por el uso de la CPU del 100%. Algunas de las estadísticas generadas mas relevantes de esta implementación son:

- **Concurrency Level:** 1500
- **Time taken for tests:** 21.558 seconds
- **Complete requests:** 1500
- **Failed requests:** 0
- **Total transferred:** 201000 bytes
- **HTML transferred:** 27000 bytes
- **Requests per second:** 69.58 [#/sec] (mean)
- **Time per request:** 21558.088 [ms] (mean)
- **Time per request:** 14.372 [ms] (mean, across all concurrent requests)
- **Transfer rate:** 9.11 [Kbytes/sec] received

3.2 Implementación `taller_2_workerpool`

Con esta implementación, el rendimiento fue extremadamente mayor. Fue posible realizar el test con la primera herramienta con 20000 conexiones concurrentes. Algunas de las estadisticas son:

- **Concurrency Level:** 20000
- **Time taken for tests:** 6.945 seconds
- **Complete requests:** 20000
- **Failed requests:** 0
- **Total transferred:** 2380000 bytes
- **HTML transferred:** 80000 bytes
- **Requests per second:** 2879.71 [/sec] (mean)
- **Time per request:** 6945.140 [ms] (mean)
- **Time per request:** 0.347 [ms] (mean, across all concurrent requests)
- **Transfer rate:** 334.65 [Kbytes/sec] received

Como se puede apreciar, a comparación de la primera implementación, las solicitudes fueron respondidas mas rápido, haciendo que no se degrade el sistema. Para evaluar la máxima cantidad de conexiones concurrentes se va a utilizar bombardier. Dicha herramienta concluyó su ejecución con las siguientes estadísticas:

- HTTP codes:
  - 1xx - 0, 2xx - 35898, 3xx - 0, 4xx - 0, 5xx - 0
  - others - 3484
- Errors:
  - dial tcp 127.0.0.1:3000: connect: cannot assign requested address - 3484
- Throughput: 0.86MB/s

Las estadísticas anteriores indican que $\approx$ 3500 solicitudes no pudieron ser atendidas debido a un error inesperado en el protocolo TCP de conexión al puerto. Esto indica que el servicio pudo atender $\approx$ 36000 usuarios concurrentes. Esta cifra puede variar en varias ejecuciones del benchmarking.

3.3 Implementación taller_2_cluster

Para esta implementación tambien fue posible llegar al máximo rango de Apache Benchmarking tool, la herramienta generó las siguientes estadisticas:

- **Concurrency Level:** 20000
- **Time taken for tests:** 7.112 seconds
- **Complete requests:** 20000
- **Failed requests:** 0
- **Total transferred:** 2940000 bytes
- **HTML transferred:** 220000 bytes
- **Requests per second:** 2812.34 [#/sec] (mean)
- **Time per request:** 7111.508 [ms] (mean)
- **Time per request:** 0.356 [ms] (mean, across all concurrent requests)
- **Transfer rate:** 403.73 [Kbytes/sec] received

Estas estadísticas son similares a las obtenidas por la implementación con *Workerpool*. En el caso de la herramienta Bombardier, estas fueron las estadísticas:

- HTTP codes:
  - 1xx - 0, 2xx - 41882, 3xx - 0, 4xx - 0, 5xx - 0
  - others - 3500
- Errors:
  - dial tcp 127.0.0.1:3000: connect: cannot assign requested address - 3500
- Throughput: 1.75MB/s

En este caso, la herramienta de Bombardier recibió una cantidad de conexiones concurrentes mas alto, ya que las cifras de ≈ 35000 no producían ningún tipo de error. Sin embargo, a las ≈ 45000 conexiones concurrentes se comenzó a perder algunas de estas conexiones por el mismo error inesperado. Para esta implementación se establece un nuevo limite de ≈ 42000 conexiones concurrentes.

3.4 Implementación taller_2_cpp

Con esta implementación, la herramienta de Apache no logró llegar al limite de conexiones concurrentes establecidas, sin embargo, la cantidad de conexiones concurrentes que soportaba fluctuaba en gran medida a lo largo de las ejecuciones del benchmarking. Algunas de las estadísticas generadas para una ejecución exitosa del benchmarking de Apache son:

- **Concurrency Level:** 14000
- **Time taken for tests:** 1.093 seconds
- **Complete requests:** 14000
- **Failed requests:** 0
- **Total transferred:** 2310000 bytes
- **HTML transferred:** 56000 bytes
- **Requests per second:** 12813.85 [#/sec] (mean)
- **Time per request:** 1092.568 [ms] (mean)
- **Time per request:** 0.078 [ms] (mean, across all concurrent requests)
- **Transfer rate:** 2064.73 [Kbytes/sec] received

Como se puede ver, el servidor con la implementación en C++ es notoriamente mas eficiente, sin embargo, en algunas de las ejecuciones no alcanzaban las $\approx$ 5000 conexiones concurrentes, mientras que en otras ejecuciones se alcanzaban cifras hasta de $\approx$ 14000 conexiones. Sin embargo, una de las características de este benchmarking es que las conexiones se cerraban por un *timeout* del servidor. De esta manera se puede ver que puede que la conexión con el socket no se haya podido establecer, pero la implementación no informa cual fue el fallo en realidad.

Chapter 4

Conclusiones

La implementación del servidor es uno de los factores mas impactantes a la hora de recibir multiples usuarios concurrentes, ya que una mala implementación o sin el uso de patrones, puede recaer en una baja del rendimiento esperado del sistema. Para un computador con las caracteristicas descritas anteriormente, la cantidad máxima de usuarios concurrentes es de ≈ 45000 **conexiones/usuarios concurrentes**.

Bibliography

[1] Rajat P. Garg and Ilya Sharapov. *Techniques for Optimizing Applications: High Performance Computing*. Sun microsystems, 2002.