

Optimización y Gestión de Recursos en Microservicios C++

Antonio Espín Herranz

Contenidos

- Optimización del rendimiento en microservicios de alto rendimiento:
 - Uso de técnicas de **gestión de memoria** eficiente (malloc/free, RAII).
 - Minimización de la **latencia** en sistemas distribuidos con C++.
- **Profiling y benchmarking** de microservicios:
 - Uso de herramientas como Valgrind, gprof o perf.
 - Estrategias para mejorar la eficiencia del CPU y la memoria.

Optimización de rendimiento en microservicios de alto rendimiento

Profiling y benchmarking de microservicios

Profiling

- Análisis del rendimiento, es una técnica que permite estudiar el comportamiento de un programa mientras se ejecuta.
- Permite identificar cuellos de botella, optimizar recursos y mejorar el rendimiento general.

¿Qué es el profiling?

- Es el proceso de **medir y analizar** aspectos internos de una aplicación como:
 - Tiempo de ejecución por función o método
 - Uso de CPU y memoria
 - Accesos a disco o red
 - Frecuencia de llamadas
 - Fugas de memoria o ciclos innecesarios
- El resultado es un **perfil de ejecución**, que te muestra qué partes del código consumen más recursos o tiempo.

¿Para qué sirve?

- Optimizar código crítico: Saber qué funciones ralentizan el sistema
- Reducir consumo de recursos: CPU, RAM, disco
- Detectar fugas de memoria
- Mejorar escalabilidad: Preparar el sistema para más carga
- Tomar decisiones informadas: Qué refactorizar, qué paralelizar, qué cachear

Benchmarking

- Es una técnica que consiste en **medir, comparar y analizar el rendimiento** de un sistema, proceso o componente frente a otros similares, con el objetivo de identificar oportunidades de mejora y adoptar las mejores prácticas.
- **Consiste en:**
 - Ejecutar pruebas de rendimiento sobre un programa, función o sistema
 - Medir métricas como tiempo de ejecución, uso de CPU, memoria, latencia, etc.
 - Comparar esos resultados con otras versiones, competidores o estándares

Tipos de benchmarking técnico

Tipo	Descripción breve
Sintético	Usa pruebas artificiales para estresar componentes
Aplicativo	Usa cargas reales simuladas (ej. peticiones HTTP)
Comparativo	Compara tu sistema con otros (competencia o versiones)
Interno	Compara entre áreas o módulos dentro de tu propio sistema

¿Para que sirve?

- Detectar cuellos de botella
- Validar mejoras de rendimiento
- Justificar decisiones tecnológicas
- Optimizar recursos (CPU, RAM, red)
- Aumentar la escalabilidad

Herramientas Linux

Herramienta	Función principal
<code>perf</code>	Análisis de CPU, eventos, y llamadas
<code>Valgrind</code>	Fugas de memoria, profiling básico
<code>gprof</code>	Perfil por función (requiere recompilación)
<code>Callgrind</code> + <code>KCachegrind</code>	Visualización de llamadas
<code>strace</code>	Trazado de llamadas al sistema

Herramientas Windows

Herramienta	Función principal
Visual Studio Profiler	Integrado, muy completo
Windows Performance Analyzer (WPA)	Análisis profundo del sistema
Intel VTune	Avanzado, multiplataforma
Process Explorer	Monitoreo en tiempo real
Very Sleepy	Profiling ligero por función

- En Windows podemos instalar los comandos de Linux en WSL (ojo utilizar **WSL 2**).

Herramientas

- **valgrind**

- Detecta fugas de memoria, errores de acceso, y profiling básico.
- <https://valgrind.org/>

- **gprof**

- Genera perfiles de tiempo de ejecución por función

- **perf** (Linux)

- Herramientas de bajo nivel para análisis de CPU y eventos.

Valgrind: instalación en WSL2

```
sudo apt update
```

```
sudo apt install -y build-essential make g++ wget
```

```
wget https://sourceware.org/pub/valgrind/valgrind-3.25.1.tar.bz2
```

```
tar -xvjf valgrind-3.25.1.tar.bz2
```

```
cd valgrind-3.25.1
```

```
./configure
```

```
make
```

```
sudo make install
```

Valgrind

- Podemos **detectar**:
 - Fugas de memoria en las clases que utilizan memoria dinámica
 - Accesos inválidos a memoria.
 - Uso incorrecto de new / delete o malloc / free
 - Consumo excesivo del Heap
- **Aplicarlo**:
 - `valgrind --tool=memcheck ./microservicio`

Gprof: instalación en WSL2

gprof

- Permite detectar:
 - Qué funciones consumen más tiempo
 - Cuántas veces se llaman
 - Relación entre funciones (quien llama a quien)
- **Aplicarlo:**
 - Compilar con la opción `-pg`:
 - `g++ -pg main.cpp -o microservicio`
 - Ejecutar el programa normalmente:
 - `./microservicio`
 - Generar el perfil:
 - `gprof microservicio gmon.out > perfil.txt`

Perf: instalación en WSL2

perf

- Permite **detectar**:
 - Uso de CPU por función o hilo
 - Llamadas al sistema (syscalls)
 - Latencia en operaciones de red o disco
 - Contención entre hilos.
- Aplicarlo:
 - **perf** record ./microservicio
 - **perf** report

Relación entre las 3 herramientas

- **PASOS:**

- 1. Simular carga con **wrk** o **curl**.
- 2. Ejecutas **valgrind** para detectar fugas.
- 3. Usas **gprof** para ver qué funciones dominan el tiempo.
- 4. Usas **perf** para ver si hay contención o latencia en llamadas externas.