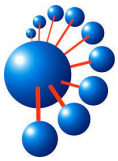
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
Y CIENCIAS DE LA COMPUTACIÓN
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE CONCEPCIÓN



Plataforma para medir el consumo energético de algoritmos

por
Diego Caripán Uribe

Memoria presentada para la obtención del título de
Ingeniero Civil Informático

Patrocinante: José Fuentes Sepúlveda

Concepción, 5 de mayo de 2022

«Agradecimientos» A todos los que me apoyaron para llegar a este momento: Mi familia, compañeros y los docentes de la universidad.

Resumen

En esta memoria se muestra el diseño e implementación de la plataforma de medición del consumo energético "*PowerTester*", que entrega resultados sobre el rendimiento energético, temporal, entre otros, de programas implementados en el lenguaje C++ en distintos medidores conectados a un servidor, con el cual un usuario interactúa.

Para esto, los medidores utilizan la tecnología de Intel *RAPL*, una tecnología que Intel incluye en ciertos chipsets, que permite la captura de métricas predefinidas propias del sistema, principalmente de la CPU. Las métricas que mas énfasis tendrán serán las de consumo energético y potencia, seguidos por otras secundarias, como el rendimiento de los niveles de caché y la duración del programa ejecutado.

Para acceder a esta información, Linux provee una herramienta de análisis de rendimiento llamada *perf*, que puede acceder a ciertas interfaces de medición del sistema, como las métricas de *RAPL*. Cada medidor utiliza esta herramienta en la ejecución del programa en C++, corriendo el programa múltiples veces y recolectando sus resultados. Una vez recolectada la información, los medidores la envían al servidor, el que se encarga de ordenarla y mostrarla mediante gráficos, reporta promedios y su correspondiente archivo CSV disponible para ser descargado y analizado de manera online.

Para validar la plataforma, se realizaron encuestas en donde los voluntarios probaron códigos previamente entregados, para luego responder preguntas sobre las métricas que se obtenían de tales códigos. Utilizando la participación de 3 voluntarios, se realizaron estos experimentos en la plataforma, con éxito, para luego proveer de su opinión. Finalmente, utilizando la escala de usabilidad de sistema *SUS*, asignamos una calificación al sistema, utilizando los comentarios de los voluntarios, y discutimos sobre los ámbitos en que se pueden mejorar, como también la sugerencia de trabajos futuros.

Índice

Índice de figuras

Índice de cuadros

1. Introducción

Hoy en día, casi todos los objetos que utilizamos dependen de energía eléctrica, recurso que cada día se vuelve más valioso por factores como el aumento de población y su uso cotidiano, el cambio climático y la creciente complejidad de la tecnología. En este último, el consumo energético es un ámbito importante, ya que el mal uso de la energía puede generar costos innecesarios y crear el riesgo de dañar dispositivos. Muchos de estos utilizan algoritmos para realizar su utilidad, y la mayor parte del consumo viene de una continua ejecución. Es por esto que el enfoque central, en el diseño de algoritmos, debe cambiar. El consumo energético de las implementaciones debe ser considerado, concepto que no se toca mucho entre programadores, pues se da prioridad a que un código sea rápido y utilice poco espacio a que utilice poca energía eléctrica. A pesar que el consumo energético depende directamente de estos factores, su medición como métrica independiente debe ser estimado, en especial en un mundo donde los dispositivos que utilizamos se vuelven más complejos, que por consecuencia, requieren un mayor consumo eléctrico.

Si queremos analizar el consumo eléctrico de programas computacionales, debemos medirlo. Para esto existen 2 grupos de herramientas. El primer grupo corresponde a modelos de estimación basados en contadores de rendimiento. Estos contadores son altamente precisos, pero dependen de la plataforma en la que están integradas. Utilizando estos, podemos estimar el consumo eléctrico de algún programa en particular. Una de las interfaces más prominentes de este grupo es la tecnología de Intel *RAPL* (Running Average Power Limit) [3]. *RAPL* provee de una estimación del consumo energético en distintos niveles (ver figura 1):

- Nivel de núcleos (todos los núcleos de un procesador)
- Nivel de paquete (todos los núcleos de un procesador, controlador de memoria, último nivel de caché, y otros componentes)
- Nivel de memoria DRAM

Estos contadores pueden ser obtenidos de registros especializados. Estos registros pueden ser accedidos directamente, o mediante la utilización de Linux *perf* o la biblioteca *PAPI*. *RAPL* ha probado ser una herramienta de medición precisa [3]. El segundo grupo corresponde a analizadores y sensores

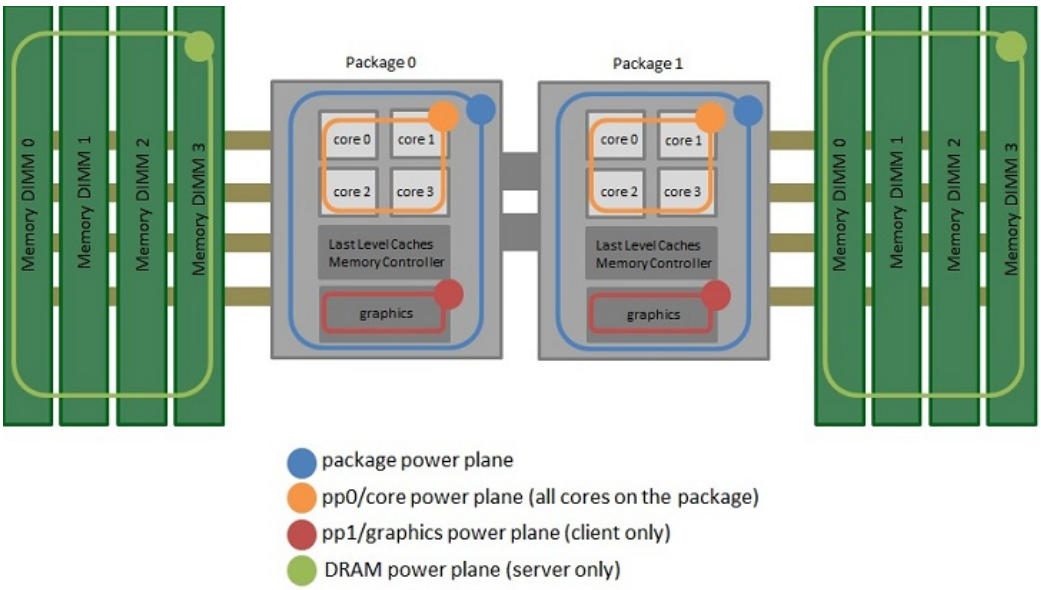


Figura 1: Representación visual[1]de los niveles de estimación de *RAPL*.

de hardware. Estos son independientes de las plataformas, ya que analizan la potencia eléctrica suministrada a la máquina, es decir, de todo el sistema computacional al que está conectado.

Sin embargo, el acceso a las herramientas de los grupos es distinta: Para el primer grupo, la herramienta depende exclusivamente de la plataforma, y para el segundo, se deben adquirir estos medidores externos, que pueden traer un costo monetario importante, además de complicaciones a la hora de medir el consumo eléctrico, como la incapacidad de aislar el consumo de un programa del sistema computacional. Utilizando un servicio web, podemos evitar que el usuario sea el que se encargue de conseguir los medidores, y permitirle obtener solo lo que desea, que es la medición del consumo energético de un programa, requiriendo solo una conexión a Internet.

De la Fuente detalla sobre el consumo energético de ciertos algoritmos, que son conocidos en el estado del arte, utilizando la interfaz de *RAPL*, que pertenece al primer grupo de herramientas de medición, y recolectando sus métricas [2]. Su método de recolección prueba ser útil para utilizar compu-

[1]Obtenido de `https://blog.chih.me/read-cpu-power-with-RAPL.html`, accedido por última vez Abril 2022

tadores como medidores. Pero su trabajo no es replicable de manera sencilla, algo necesario para la implementación de múltiple medidores.

1.1. Objetivo General

La creación de una plataforma de medición de códigos C++, que pueda ser accedida por cualquier usuario con acceso a Internet, que, teniendo cierto conocimiento sobre consumo energético y programación de C++, pueda obtener métricas sobre la ejecución de alguna implementación, dando conclusiones sólidas sobre su ejecución en distintas máquinas medidoras.

1.2. Objetivos Específicos

Esta memoria busca:

- Implementar un código para la medición de consumo energético de programas en C++ utilizando computadores nodo como medidores.
- Implementar un servicio Web tipo cliente-servidor para la medición de diferentes métricas, privilegiando las de consumo energético, utilizando los medidores ya mencionados.
- Evaluar experimentalmente la usabilidad de la plataforma.
- Publicar el material generado, para su posterior uso en la escalabilidad de la plataforma, en repositorios de libre acceso.

1.3. Limitaciones

Como esta es la primera iteración de la plataforma, su usabilidad será limitada, que pueden ser abordadas en futuras versiones de la plataforma.

2. Trabajos relacionados

Programadores y consumo energético

Los programadores podemos aprender un gran rango de lenguajes y metodologías de programación, pero muy raramente tocamos la eficiencia energética. El uso masivo de la computación en la nube y dispositivos móviles requieren un aumento en la atención sobre el consumo energético. Pang *et al.* [4] realizaron un estudio sobre una encuesta de 13 preguntas para estimar el grado de conocimiento de estas prácticas. Utilizando una red social, lograron recolectar 121 respuestas, en las cuales los encuestados se identificaban como programadores. Hay que destacar que 37 de los 121 encuestados mencionaron utilizar C o C++, lenguaje que puede ser medido utilizando la plataforma desarrollada en esta memoria de título, dando una justificación a la necesidad de un trabajo futuro que aumente la cantidad de lenguajes de programación aceptados.

La encuesta entregó los siguientes resultados importantes:

- Los programadores no se dan cuenta, o no le es solicitado abordar la eficiencia energética, sugiriendo la falta de atención por prioridades.
- Los usuarios finales raramente se quejan sobre problemas de consumo energético en software.
- **Los programadores tienen poco o nulo conocimiento sobre la medición de consumo energético y de cómo hacerlo.**
- **La mayoría de los programadores piensan en hardware cuando se les pregunta sobre consumo energético.**
- El conocimiento sobre el consumo energético es más consistente sobre dispositivos móviles.
- Muchos programadores no conocen los principales factores que afectan el consumo energético de software.

Los ítems destacados en negrita son problemáticas que la plataforma que se desarrolló puede resolver, incentivando la importancia de software en el consumo energético, y dando una herramienta de medición que un programador podría aprender a utilizar.

La elección afecta al consumo

Cuando se trata del usuario final, muy pocos toman en cuenta el consumo energético de una aplicación de software, generalmente solo ocurre cuando el dispositivo que lo ejecuta tiene una capacidad limitada de energía, como un *smartphone*. Esto indica que la elección del usuario de qué aplicación utilizar tiene un impacto directo en el consumo energético y vida de la batería de un dispositivo móvil. Zhang *et al.* [6] realizaron un estudio sobre el consumo energético de diversas aplicaciones, en 3 tipos de escenarios, además de un escenario donde el sistema esta en reposo:

- Editores de texto.
- Clientes de correo electrónico.
- Reproductores de música.

Entre sus conclusiones se destacan:

- Aplicaciones basados en Web son menos eficientes que sus contrapartes offline.
- Interfaces (GUI) que se actualizan continuamente consumen una cantidad de energía significativa.
- Futuras aplicaciones deberían documentar el impacto energético de sus distintas características e incluir la capacidad de elegir un nivel de consumo, como lo hacen los sistemas operativos.
- La creación de un sistema de rating estandarizado *Software Application Energy Consumption Rating*, para que usuarios finales, sin conocimiento, conozcan el rendimiento energético de aplicaciones.

De las tres categorías, las aplicaciones que más energía consumían, en promedio, eran las de edición de texto. En esta categoría, se encontraban 3 aplicaciones: Gedit, LibreOffice y Google Docs. Google Docs utilizaba mucha más energía que las otras, puesto que tiene la característica de guardado automático, lo que hace que la sincronización constante utilice mucha más energía. Cabe señalar que la interfaz de la plataforma no posee una actualización continua y las entradas de texto no admiten formatos complejos.

Formas de medición

Por otro lado, también se han mencionado la existencia de otras formas de medición. Los medidores basados en software son precisos al recolectar métricas a nivel de aplicación, pero dependen de la plataforma. Intel *RAPL* solo esta disponible en ciertos chipsets, a partir del lanzamiento de la arquitectura Sandy Bridge, y su competencia, AMD, no posee una interfaz lo suficientemente madura para compararse con *RAPL*.

PowerAPI[2] proporciona un script llamado Jouleit, que utiliza un método similar de recolección de datos empleado en esta memoria. Sin embargo, PowerAPI busca proporcionar herramientas para la recolección de mediciones, que son utilizados en una formula, definida por el usuario, para estimar el consumo energético de una maquina. Esta formula utiliza el toolkit de PowerAPI.

Los medidores físicos se deben conectar directamente, como Watts Up Pro[3], pero es difícil distinguir el consumo de un programa del consumo de la máquina entera.

Finalmente, existe una categoría emergente: el uso de inteligencia artificial. Chowdhury *et al.* [1] proponen el uso de un modelo de inteligencia artificial *GreenScaler*, que busca estimar el consumo energético de aplicaciones móviles, mediante el uso de datos aleatorios y reales. Este tipo de medición pretende disminuir el costoso trabajo de extraer data de pruebas que requieren ser medidos durante sus ejecuciones, o de pruebas manualmente escritas, que consumen tiempo. Sin embargo, para entrenar la inteligencia artificial, se necesitan datos reales, que, irónicamente, se obtiene de los otros tipos de medidores. La plataforma que se desarrolló en esta memoria tiene el potencial de apoyar este tipo de categoría, puesto que los datos recolectados son entregados en formato CSV. Una inteligencia artificial podría analizar cientos de estos CSV y entrenar para lograr una exactitud rigurosa e incluso ser agregado a esta plataforma como anexo en un trabajo futuro.

[2] `https://powerapi-ng.github.io/introduction.html`, accedido por última vez Abril, 2022.

[3] `https://www.amazon.com/Watts-Pro-Electricity-Consumption-Meter/dp/B000CSWW92`, accedido por última vez en Abril, 2022.

Medición sistemática

La base de esta memoria viene del trabajo de De la Fuente [2], que utilizó la interfaz de *RAPL*, a través de *perf*, para medir algoritmos de búsqueda y ordenamiento. La plataforma utiliza el mismo método que él utilizó para recolectar métricas, que es a través de un script de Bash. De la Fuente demuestra que el consumo energético no siempre, pero generalmente, depende directamente del tiempo de ejecución de un algoritmo. Mostró ciertas ejecuciones que, ha pesar de ser rápidas, utilizaban más energía que algoritmos similares. Por otro lado, también concluye que la cantidad de datos de entrada puede aumentar el consumo energético, como es la implementación del algoritmo. Este último punto está marcado por la comparación de las versiones iterativas y recursivas de ciertos algoritmos, como Merge Sort.

Como se mencionó anteriormente, el trabajo de De la Fuente no es fácilmente replicable, debido a que es necesario tener equipamiento similar, una máquina que tenga un chipset de Intel, además de saber utilizar Linux, y con esto, *perf*. La plataforma busca aliviar esto, abstrayendo estas necesidades a los usuarios, para obtener solo la información deseada: El rendimiento y el consumo energético de un código, código que puede ser utilizado como parte de aplicaciones más grandes, como la creación de software o aplicaciones móviles, o necesidades más simples, como la recolección de datos para futuros trabajos relacionados.

3. Diseño e Implementación

3.1. Modelo principal

El modelo que utilizará la plataforma es de Cliente-Servidor, utilizando las prácticas de AJAX. El cliente interactúa con una aplicación web que se comunica con el servidor, sin interferir en la interfaz, permitiendo que su contenido cambie dinámicamente. El uso prominente de estas prácticas de interfaces reactivas lo vuelve particularmente atractivo (como Facebook y Netflix). Además el servidor se conecta con medidores, a los cuales les envía códigos provistos por los usuarios. Por su parte, los medidores toman las métricas de rendimiento, las que envían de regreso al servidor. (Ver figura 2 para un ejemplo del modelo utilizado)

Este modelo requiere que existan 2 implementaciones distintas: Uno para el cliente y otro para el servidor.

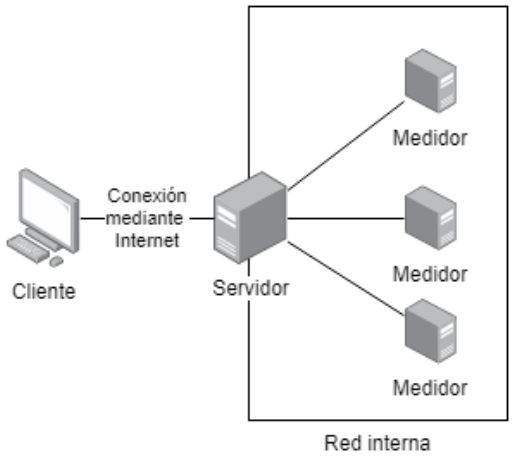


Figura 2: Diagrama del modelo principal de la plataforma

3.2. Lenguajes y bibliotecas

Los lenguajes de programación utilizados en la memoria son **Python** y **JavaScript**, específicamente la extensión de React, **JSX**, y en menor grado, **Bash**.

La implementación del cliente utiliza **JavaScript**, mientras que la del servidor, **Python**.

En Python, la principal biblioteca utilizada es **Flask**, un microframework utilizado para el desarrollo de páginas y aplicaciones web. Otras bibliotecas notables incluidas son:

- `MatPlotLib` (Biblioteca de generación de gráficos)
- `Pandas` (Biblioteca de manipulación de datos y análisis)
- `NumPy` (Biblioteca de estructuras y métodos matemáticos)
- `threading` (Biblioteca estándar para el uso de hebras)
- `subprocess` (Biblioteca estándar para el control de procesos del sistema)
- `socket` (Biblioteca estándar para el uso de *sockets*, una estructura de software que permite el uso de computadores, como nodos, en redes de computadores)

Por otro lado, JavaScript utiliza principalmente la librería de React, junto con sus librerías estándar.

Bash se utiliza en el script de medición de métricas, que utilizó De la Fuente en su memoria, haciéndole uso en los medidores, junto con su uso en invocaciones de Python.

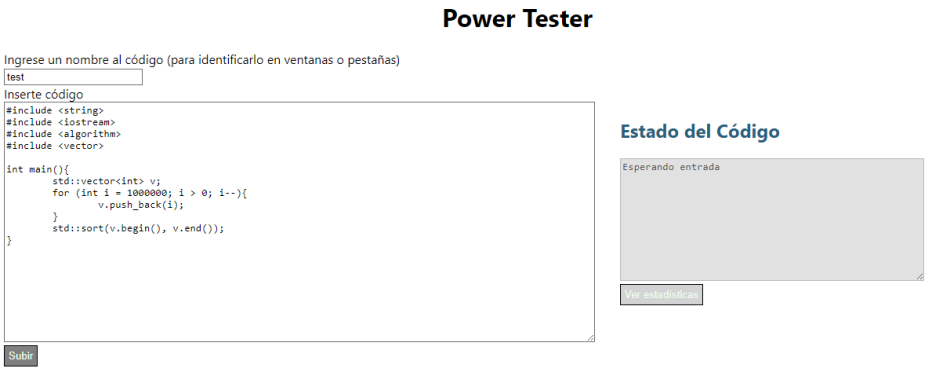


Figura 3: Captura de pantalla de la página principal. La entrada de texto tiene un código listo para ser medido.

3.3. Cliente

Del lado del cliente, la interfaz esta diseñada utilizando la librería de React de JavaScript. En ella, un formulario recibe el código a medir, para luego recibir una confirmación por parte del servidor en el recuadro "Estado de Código"(ver figura 3). En este recuadro aparece el estado del código cuando llega al servidor. El servidor lo compila, reporta cualquier error de compilación en este cuadro y descarta el código. Si no hay errores, el servidor responde con "Listo". Cuando esto ocurre, el botón "Ver estadísticas"se activa (ver figura 4), mostrando una tabla de promedio de varias métricas (ver figura 5), y gráficas que muestran los datos visualmente. (ver figuras 6 y 7).

La interfaz interactúa con el servidor mediante el uso de solicitudes HTTP, usando POST cuando envía el código, y GET, para obtener cualquier otro dato (como el estado del código).

3.4. Servidor

Por otra parte, el servidor recibe tales solicitudes, y los procesa, dependiendo del patrón URL que utilizan.

La biblioteca de Flask permite el procesamiento dependiente de rutas de URL. Para contexto, las rutas de URL son los métodos a que un cierto patrón llegan. Un patrón es un set de caracteres ordenados que están modelados sobre una URL base, y se identifican usando una barra (`/`). Dependiendo del patrón al que llegan, pueden obtener distinta información.

Figura 4: Captura de pantalla de la página principal. El botón “Ver estadísticas” está habilitado.

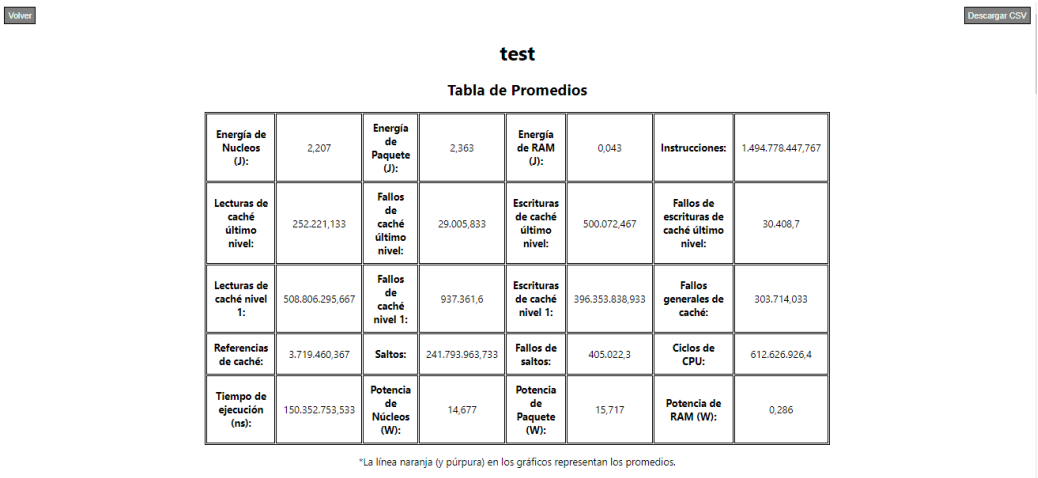


Figura 5: Captura de pantalla de la segunda página. Esta página aparece cuando se presiona “Ver estadísticas”.

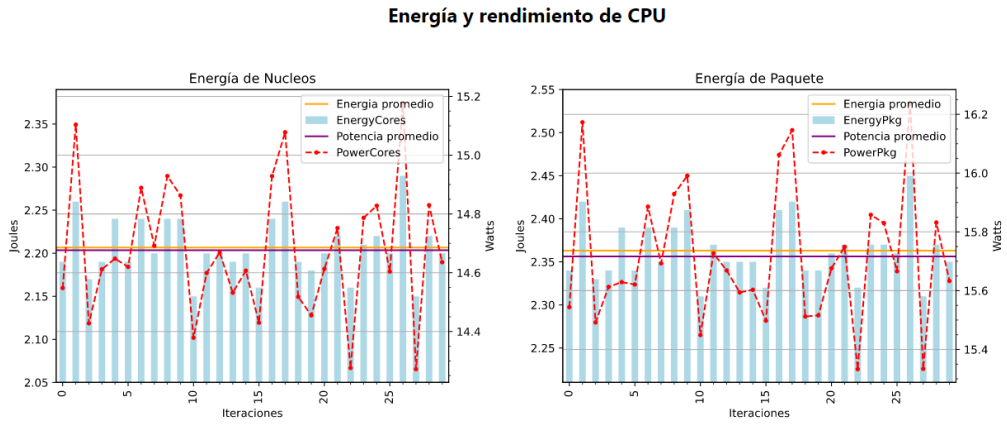


Figura 6: Continuación de captura de pantalla de la segunda página. Se muestran los gráficos relacionados con consumo energético.

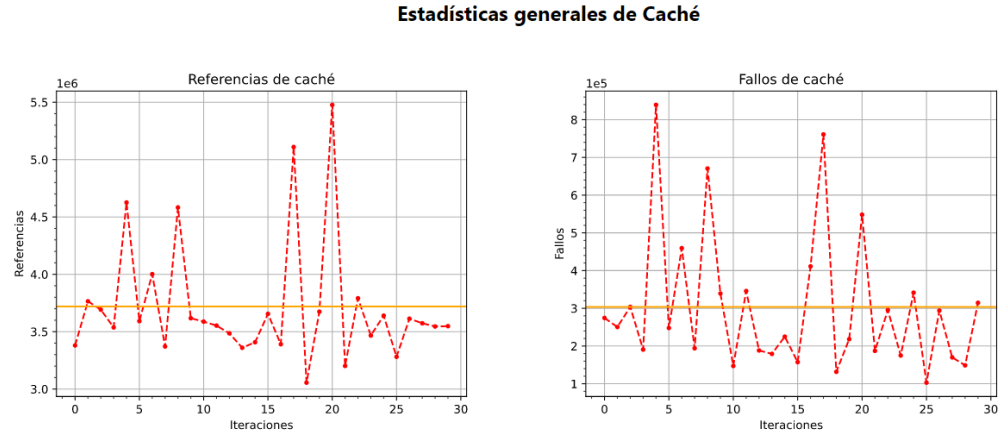


Figura 7: Continuación de la captura de pantalla de la segunda página. Se muestran más gráficos, relacionados con caché.

Como ejemplo, YouTube posee un patrón que muchos utilizamos `/watch`, sobre la URL `www.youtube.com`. Sin embargo, este patrón no da información válida si no esta acompañado por un parámetro de patrón, que, en este ejemplo, es representado como `?v=[identificador único de vídeo]`, creando la URL completa:
`www.youtube.com/watch?v=[identificador único de vídeo]`,
que muestra un vídeo válido.

La plataforma no utiliza parámetros de patrón, pero sí utiliza patrones. Los siguientes patrones son los que utilizan para su interacción con el Cliente:

- `/sendcode`: Captura el código recibido del Cliente y asigna un código de identificación único.
- `/checkstatus/<code>`: Utilizando la identificación, muestra el estado del código recibido.
- `/<code>/mean`: Calcula los promedios de las métricas generadas por el código identificado. Los envía hacia el Cliente en formato JSON.

Cuando un patrón es correcto, el servidor realiza distintas acciones relacionadas con el patrón recibido.

El servidor, antes de iniciar su recepción de solicitudes HTTP, inicializa un método que se encarga de supervisar la cola de códigos que recibe. Debido a la naturaleza de los servicios Web, muchas solicitudes pueden llegar rápidamente, por lo que existe la problemática de recibir muchos códigos y querer ser servidos al mismo tiempo. Esto no es posible, puesto que cada código puede tomar segundos en ser medido. Para ello, se implementó el método `queue_manager` que se encarga que cada código recibido entre a una cola, de las cuales van siendo servidos uno a uno, evitando así la medición concurrente de códigos, puesto que esto puede afectar en las métricas recolectadas por los medidores, y por el uso concurrente de puertos. Este método es asincrónico, es decir, siempre estará revisando el estado de la cola, y es gestionada por Flask, mediante el uso de una hebra *daemon*.

Cuando un código sale de la cola, se invoca otro método implementado `slave_serve`. Este método es encargado de comunicarse con los medidores, enviando códigos a medir y recibiendo los resultados. Esto lo hace mediante el uso de sockets, puntos de comunicación entre computadores, que están unidos a un puerto del sistema. Distintos programas pueden utilizar distintos puertos, pero solo un programa puede estar utilizando un puerto a la vez.

La conexión con los medidores se hace utilizando los sockets con los puertos 50000, que se utiliza para el envió de códigos, y puertos 60000, que se utiliza para recibir las mediciones. El método hace el uso de hebras para enviar y recibir a todos los medidores que intenten conectarse con el servidor.

La figura 8 representa el flujo de comunicación entre el servidor y los medidores.

Una vez recibidas las mediciones, estas son almacenadas en un archivo CSV. Posteriormente, para el procesamiento de las mediciones se utilizó la biblioteca `Pandas`, de la cual también se obtuvieron representaciones gráficas de los resultados. Siendo más especifico, `Pandas` posee un estructura de datos llamada `DataFrame` que permite asignarle datos utilizando un archivo CSV. Este objeto posee el método `plot()` que utiliza la biblioteca de `MatPlotLib` para la generación de gráficos de los datos presentes en el objeto `DataFrame`. Así cada columna del archivo CSV es graficada, excepto las columnas de energía, que comparten gráficos junto con las columnas de potencia.

3.5. Medidor

En los medidores, existen 2 archivos importantes, `slave.py` y `measurescript.sh`.

`slave.py` es el código en Python que los medidores están ejecutando continuamente. Se encarga de abrir los sockets con los puertos mencionados en el servidor, e intenta crear una conexión con este, ya que el servidor solo está abierto a conexiones cuando un código requiere ser medido. Cuando recibe un código, lo hace desde el puerto 50000 y lo almacena en un archivo temporal, para posteriormente compilarlo. Una vez compilado, el medidor cierra la conexión por el puerto 50000, debido a que puede afectar negativamente en las mediciones. Luego, utilizando `measurescript.sh` y el ejecutable de la compilación, procede a recolectar las métricas de interés (energía, instrucciones, caché, entre otros). Cuando las métricas de 30 ejecuciones han sido recolectadas, estás son almacenadas en un archivo CSV, y son enviadas al servidor utilizando el puerto 60000. Este archivo CSV es generado por *perf*, por lo que su creación no afecta la medición del programa compilado. Otros procesos externos pueden afectar negativamente la medición, por lo que se recomienda tener solo los procesos vitales activos (como *systemd*, un proceso que provee diversos componentes importantes de Linux, o *wininit* en Windows). La razón de esta forma de conexión es simple: Como el servidor no conoce quienes son los medidores, solo espera por conexiones entrantes de

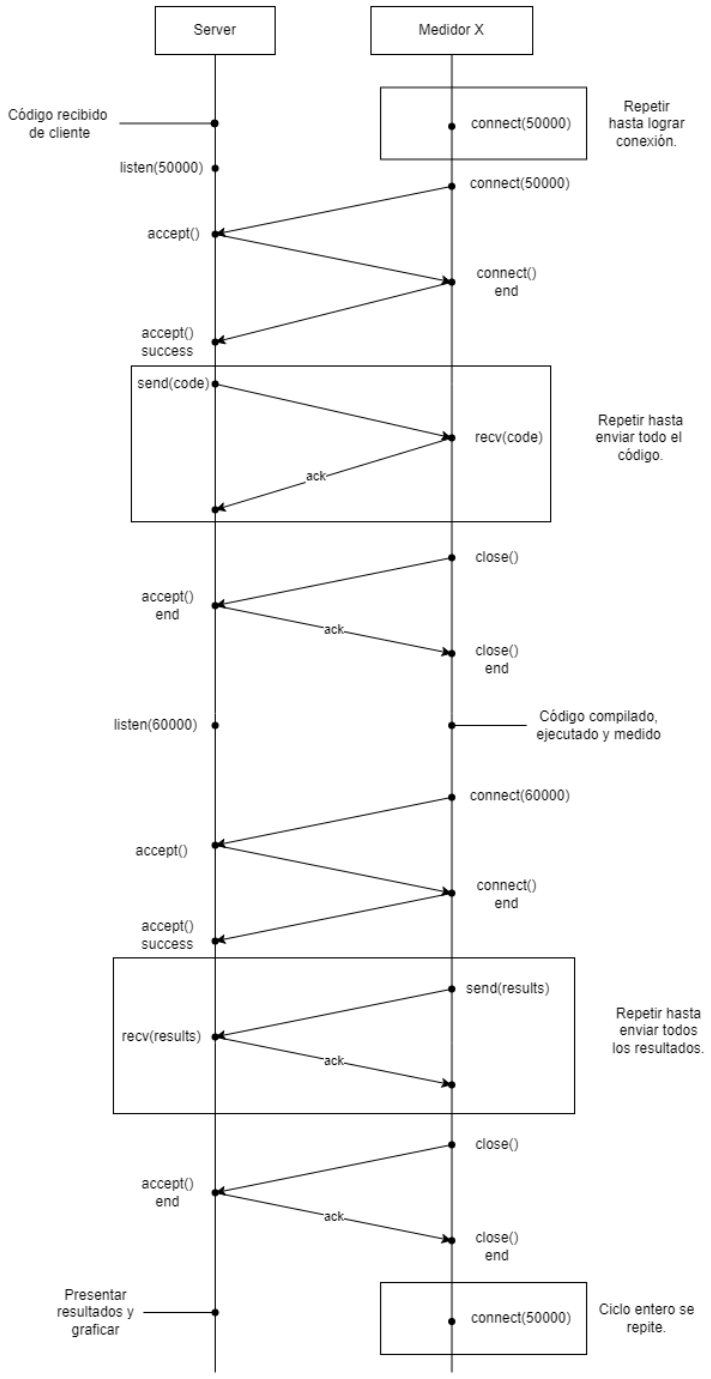


Figura 8: Diagrama de comunicación entre servidor y un medidor X.

estos. Esto permite que cualquier computadora unida a la red interna pueda tomar el rol de medidor con solo ejecutar `slave.py`, `measurescript.sh` y tener los correspondientes puertos disponibles a través del contra fuegos. Esta característica permite la fácil escalabilidad del número de medidores.

`measurescript.sh` es un script en Bash, una versión modificada de la que De la Fuente utilizó, que automatiza la ejecución de la herramienta *perf* de Linux, utilizándolo para capturar las métricas y darles formato en un archivo CSV. Es aquí donde la tecnología de Intel *RAPL* interactúa con *perf*, permitiendo la captura de las métricas relacionadas con energía.

Finalmente, el medidor se conecta con el servidor, utilizando el puerto 60000, envía el archivo CSV y reinicia el ciclo de espera por un código.

4. Experimentación y resultados

4.1. Método y descripción de algoritmos

Para probar el diseño de la plataforma, se desplegó un prototipo en un servidor físico disponible en la red interna de la universidad. La usabilidad de la plataforma fue probada de la siguiente forma:

1. Un usuario sube al servidor un código que desea medir, a través de la interfaz del servicio web.
2. El servidor recibe el código y lo distribuye a sus medidores.
3. Los medidores prueban el código, obtienen métricas y las envían de vuelta al servidor.
4. El servidor ordena las métricas, obtiene promedios y gráfica los resultados
5. El usuario puede descargar el CSV de los resultados y puede observar las gráficas generadas en el servicio web.

Entre las pruebas, se varía el código que se envía para observar si la plataforma cumple con su trabajo correctamente en distintos tipos de código. Para hacer las pruebas, voluntarios utilizan códigos que han sido proporcionados (ver Tabla 1), la mayoría utilizados por De la Fuente, para luego contestar una encuesta. Los códigos proporcionados están divididos en 3 categorías: Ordenamiento, Búsqueda y Edición.

Los voluntarios realizan una primera encuesta[4] en donde se pide utilizar estos códigos en la plataforma desplegada en la URL `http://keira.inf.udec.cl:8080`, y luego responder algunas preguntas relacionadas con comparaciones entre códigos, separados por en las tres categorías.

En la sección de Ordenamiento, le pedimos a los voluntarios que ordenen de menor a mayor consumo de energía distintas implementaciones de algoritmos de ordenamiento, tomando en cuenta los resultados de las métricas preguntadas.

En la sección de Búsqueda, le pedimos a los voluntarios hacer la comparación entre una implementación recursiva y una iterativa de un algoritmo de búsqueda, utilizando las métricas proporcionadas por la plataforma.

[4] `https://ncv.microsoft.com/XhQ7zMJW3i`, accedido por última vez en abril del 2022

Cuadro 1: Códigos utilizados* para probar la plataforma.

Categoría	Códigos	Descripción
Ordenamiento	`stdsort.cpp`	Implementación del algoritmo ordenamiento de la Biblioteca estándar (`sort()`).
	`qsort.cpp`	Implementación del algoritmo del ordenamiento Quick sort, de la Biblioteca estándar (`qsort()`).
	`mergesort.cpp`	Implementación del algoritmo de ordenamiento Merge sort.
	`heapsort.cpp`	Implementación del algoritmo de ordenamiento Heap sort
Búsqueda	`bsiterative.cpp`	Implementación del algoritmo de búsqueda binaria iterativa.
	`bsrecursive.cpp`	Implementación del algoritmo de búsqueda binaria recursiva.
Edición	`replace.cpp`	Implementación de un algoritmo de edición de archivos simple.

* Disponibles en las carpetas de `https://github.com/diego-caripan/power-tester/tree/main/Ejemplos`.

Finalmente, en la sección de Edición, le pedimos a los voluntarios comentar sobre métricas de la ejecución de un código que utiliza el manejo de archivos, utilizando 2 archivos de entrada similares.

4.2. Descripción y análisis de las encuestas

4.2.1. Primera encuesta: Testing

Las siguientes preguntas fueron utilizados en la primera encuesta, separados en las secciones mencionadas, y enumeradas:

- Sección de algoritmos de ordenamiento:
  - Ordenar los algoritmos por su eficiencia considerando las siguientes métricas:

    P1 Energía de Núcleos.

    P2 Número de instrucciones.

    P3 Fallos de caché generales.

  P4 ¿Puede descargar y abrir el CSV?

- Sección de algoritmos de búsqueda:
  - Determinar cuál versión del algoritmo de búsqueda binaria posee mejor rendimiento considerando las siguientes métricas:

    P5 Potencia de Núcleos.

    P6 Escrituras en la caché nivel 1.

    P7 Escrituras en la caché del último nivel.

  P8 Utilizando los gráficos, ¿Se observa una diferencia aparente?

- Sección de algoritmo de edición:

  P9 ¿Se observa un aumento considerable de las métricas (entre los dos archivos leídos)?

  P10 ¿Nota alguna diferencia con los algoritmos de las otras secciones?

  P11 ¿Que diferencias?[5]

  P12 Usando los gráficos y estadísticas, ¿qué tan grande estima que es el archivo 1 comparado con el archivo 2?

[5]Esta pregunta solo aparece si la anterior se responde positivamente.

Para lograr que las pruebas sean los más cercanos a la realidad, se hacen las pruebas de los códigos utilizando archivos de entrada. Estos archivos también están divididos por las categorías mencionadas: Ordenamiento utiliza un archivo de 10.500.000 valores desordenados, con un tamaño de 79.5 MB. Búsqueda utiliza una copia del archivo anterior, pero ordenado de menor a mayor. Edición utiliza 2 archivos, de 20 MB y 100 MB respectivamente, de caracteres ASCII aleatorios.

Estas pruebas buscan validar la consistencia de la información entregada por la plataforma, esperando que todo los voluntarios respondan la primera encuesta de la misma manera. Luego, se les pide responder una segunda encuesta[6]. Esta encuesta hace preguntas sobre la plataforma, que son utilizadas para calcular una calificación en la escala de usabilidad del sistema (*SUS*).

En las encuestas, estando disponibles por una semana, solo se obtuvieron 3 respuestas. En la primera encuesta, la mayoría de estás respondieron de manera similar, por lo que se puede inferir que la medición de la máquina es consistente (ver tabla 2). En la mayoría de las preguntas, la respuestas populares son la respuestas esperadas.

4.2.2. Segunda encuesta: Usabilidad

Por otro lado, en la segunda encuesta, las respuestas en las preguntas de usabilidad fueron menos consistentes (ver tabla 3). Para calcular la puntuación SUS, las opciones de las preguntas se le son asignadas valores 1 al 5, donde 1 es "Muy en desacuerdo", y 5, "Muy de acuerdo". Los valores obtenidos de las preguntas de numeración impar se les resta 1, mientras que las de numeración par, 5. Finalmente, los valores de todas las preguntas son sumadas para obtener la puntuación. Esta puntuación es única por cada respuesta, por lo que se debe calcular un promedio de todas las respuestas, para tener una calificación certera. Cabe destacar que cuando todas las preguntas son "Neutral", la puntuación es 50, y 68 es el valor donde el sistema se considera marginal, que es por debajo del valor promedio. Con una puntuación promedio de 56.7 (ver tabla 3), la plataforma demuestra tener cierto grado de usabilidad, pero necesita ser mejorado en futuras iteraciones. Utilizando los comentarios de los participantes, una forma de volver la plataforma mas útil, es implementando la capacidad de comparar códigos, sin tener que enviar códigos de uno en uno. También, agregar la capacidad de recibir datos de

[6] `https://ncv.microsoft.com/vioJLGbgjd`, accedido por última vez en abril del 2022

Cuadro 2: Resultados de la encuesta 1.

Pregunta	Respuesta más popular	Porcentaje de respuestas similares a la esperada
P1	1. qsort 2. stdsort 3. mergesort 4. heapsort	67 %
P2	1. qsort 2. stdsort 3. heapsort 4. mergesort	67 %
P3	1. stdsort 2. qsort 3. mergesort 4. heapsort	67 %
P4	Si	100 %
P5	Versión iterativa	100 %
P6	Versión iterativa	100 %
P7	Versión iterativa	100 %
P8	"Si, se observa una diferencia"	100 %
P9	Si	100 %
P10	No	67 %
P11[+]	"Realiza mas ciclos de CPU"	33 %
P12	"Aproximadamente 5 veces mas grande"	100 %

[+] Esta pregunta solo es visible si la anterior (pregunta 10) responde Si.

entrada proporcionados por el usuario, para poder analizar la escalabilidad de los algoritmos con respecto a la cantidad de datos de entrada. Este último punto es uno de las ideas centrales de trabajos futuros.

Cuadro 3: Preguntas y respuestas de la encuesta 2 (SUS).

Pregunta	Participante 1	Participante 2	Participante 3
Creo que me gustaría usar esta plataforma con frecuencia	De acuerdo	Neutral	Neutral
Encontré esta plataforma innecesariamente compleja	Neutral	En desacuerdo	De acuerdo
Pensé que esta plataforma es fácil de usar	Neutral	Neutral	Muy en desacuerdo
Creo que necesitaría asistencia para usar esta plataforma	De acuerdo	De acuerdo	Neutral
Encontré que las funciones de la plataforma están bien integradas	De acuerdo	Muy de acuerdo	Neutral
Encuentro que hay mucha inconsistencia en la plataforma	En desacuerdo	Muy en desacuerdo	En desacuerdo

Continuación de cuadro 3.

Pregunta	Participante 1	Participante 2	Participante 3
Creo que personas aprenderían a usar esta plataforma rápidamente	De acuerdo	Neutral	En desacuerdo
Encontré esta plataforma muy incómoda/difícil de utilizar	En desacuerdo	Neutral	Neutral
Me siento cómodo usando la plataforma	Neutral	De acuerdo	Neutral
Necesité aprender de muchas otras cosas para poder utilizar la plataforma	En desacuerdo	Muy en desacuerdo	Muy en desacuerdo
Puntuación SUS	**55**	**67.5**	**47.5**

5. Conclusiones y trabajo futuro

En esta memoria de título se diseño e implementó una plataforma para la medición del consumo energético de códigos escritos en C++, utilizando la tecnología de Intel *RAPL*. La plataforma fue validada utilizando la escala de usabilidad de software (SUS), y se determinó que posee potencial para escalar en tamaño y complejidad, en especial, en la cantidad de medidores que se pueden utilizar. Esta plataforma servirá como base para futuros desarrollos en la medición sistemática del consumo energético de algoritmos y estructuras de datos.

Uno de los topes de esta memoria fue su validad, que sólo alcanzó a considerar 3 personas. Para una validación mas certera, se necesita hacer una encuesta que posea mayor duración y alcance. Además, no se preguntaron por perfiles y otras preguntas que ayudarían a crear más conclusiones con las encuestas, debido a que fueron utilizados para verificar la consistencia de las respuestas, y la funcionalidad de la plataforma. Por otro lado, la plataforma también posee características que deben ser mejoradas, las que pueden ser abordadas en trabajos futuros, principalmente a nivel de seguridad. Tanto el servidor como los medidores están en una red privada, pero el servidor también es visible a través de Internet, por lo que es importante hacer un chequeo de seguridad a los códigos que reciben, para evitar alguna acción maliciosa que pueda comprometer el servidor y su información.

Otro punto que se puede tomar en trabajos futuros es el diseño de la interfaz de usuario. Como el objetivo de esta memoria era crear una plataforma que utilizará medidores, el diseño de la interfaz es simple y minimalista, pero no es visualmente atractivo. Además, una funcionalidad de poder comparar códigos, como fue mencionado anteriormente, requeriría un nuevo diseño.

Como último punto, el soporte de otro lenguajes de programación también ayudaría en la usabilidad de la plataforma, puesto que, a pesar de que C++ es uno de los lenguajes más eficientes energéticamente [5], el estudio de Pang *et al.* [4] muestra que es uno de los lenguajes de programación menos usados, siendo superado por C#, JavaScript y Python.

Referencias

[1] Shaiful Chowdhury, Stephanie Borle, Stephen Romansky, and Abram Hindle. Greenscaler: Training software energy models with automatic test generation. *Empirical Softw. Engg.*, 24(4):1649–1692, aug 2019.

[2] Leonardo de la Fuente. Medición sistemática del consumo energético de algoritmos de ordenamiento y búsqueda. Master's thesis, Universidad de Concepción, 2021.

[3] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K. Nurminen, and Zhonghong Ou. Rapl in action: Experiences in using rapl for power measurements. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 3(2), mar 2018.

[4] Candy Pang, Abram Hindle, Bram Adams, and Ahmed E. Hassan. What do programmers know about software energy consumption? *IEEE Software*, 33(3):83–89, 2016.

[5] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. Energy efficiency across programming languages: How do energy, time, and memory relate? In *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*, SLE 2017, page 256–267, New York, NY, USA, 2017. Association for Computing Machinery.

[6] C. Zhang, A. Hindle, and D. M. German. The impact of user choice on energy consumption. *IEEE Software*, 31(03):69–75, may 2014.