

# Utilizando MPI con información recolectada de alarmas

Luna Damián, Massaccisi Daniel, Morales Peña Pablo, Romero Federico.

<sup>1</sup>Universidad Nacional de La Matanza,  
Departamento de Ingeniería e Investigaciones Tecnológicas,  
Florencio Varela 1903 - San Justo, Argentina  
damian.daniel.luna@gmail.com  
massaccisidaniel@gmail.com  
pablogabrielmp@gmail.com  
fedeaandresromero@gmail.com

**Resumen.** En el presente trabajo se muestra un entorno de procesamiento distribuido de información entre dispositivos que corren el sistema operativo Android. Se utiliza una variante de MPI que permite el intercambio de información en un esquema cliente-servidor, analizando la información reunida por el Molestador. Se busca estudiar la calidad del sueño de los usuarios, y analizar de esta manera patrones que permiten afirmar si el descanso entre quienes usan Molestador es correcto.

**Palabras claves:** Molestador, MPI, openMPI, MPICH, Android.

## 1 Introducción

Hoy en día, la sociedad permanece permanentemente conectada a Internet mediante una variedad enorme de dispositivos. Los mismos proporcionan un catálogo enorme de información disponible. A lo largo de los años, dichos dispositivos comenzaron a tomar un rol fundamental en el día a día de las personas, al punto de ser utilizados por primera vez al despertar, y por última vez justo antes de dormir.

Este último suceso puede provocar trastornos indeseados en la calidad del sueño de las personas. Numerosos estudios científicos aseguran que, para poder descansar correctamente durante la noche, se recomienda no menos de 8 horas seguidas de sueño. Dado el uso de dispositivos móviles hasta momentos antes de dormir, la calidad del descanso nocturno puede verse afectado. Esto se debe, en parte, al estímulo constante generado al cerebro por la pantalla del aparato utilizado.

Teniendo en cuenta la situación explicada anteriormente, y dada la funcionalidad ofrecida por el proyecto presentado, se pueden utilizar datos provenientes de los distintos Molestadores para analizar patrones de sueño entre las personas que lo utilizan. De esta manera, se pueden tomar datos más certeros para analizar cómo es el descanso de los propietarios. Todos los Molestadores se encuentran en puntos

distantes entre sí, por lo que es necesaria una solución que permita el procesamiento masivo de los datos de forma sincronizada, sin estar conectados físicamente a la misma red.

Cada Molestador se encuentra conectado a un dispositivo Android, en los cuales se puede implementar una plataforma de computación distribuida. MPI y openMPI son los principales candidatos para ser utilizados. Características propias de la arquitectura del sistema operativo Android hacen que MPI y openMPI no sean del todo compatibles, por lo tanto, determinadas rutinas no son funcionales. Esto obliga a tener que usar una tecnología distinta. [2]

MPICH (High-Performance Portable MPI) es una implementación libre y gratuita del estándar MPI, compatible con las bibliotecas utilizadas en Android. Será la tecnología a usar en este proyecto.

## 2 Desarrollo

Para permitir el trabajo distribuido, dentro de la red de dispositivos Android, se debe designar un servidor y un número de clientes que se irán conectando. Las tareas principales del servidor consisten en admitir las conexiones provenientes de las IP conectadas en la red, e inicializar la conexión segura para el intercambio de datos e instrucciones. El cliente iniciará la conexión, y al recibir la respuesta por parte del servidor, quedará a la espera de información para realizar el procesamiento, devolviendo el resultado final al servidor una vez finalizado.

MPICH distribuye el código en dos secciones: [1]

1. **Process Manager:** es la sección encargada de inicializar el estado inicial de cada proceso, asignando identificadores, prioridades, peso del proceso, entre otros. Luego, se encarga de crear un socket de conexión con el servidor y abrirlo. Una vez realizada esta tarea, su funcionalidad quedará limitada a recibir, enviar y decodificar información y comandos para mantener la conexión cliente-servidor.
2. **Process Manager Interface:** se encarga de convertir información de alto nivel, a información en formato binario, capaz de ser enviada y recibida a través de sockets. Ofrece comandos para inicializar la comunicación, métodos de comunicación uno a uno o uno a muchos. Ofrece métodos de sincronización mediante semáforos para evitar problemas de acceso concurrente a la memoria del dispositivo.

Para la implementación de MPICH (o bien MPICH2) en Android, se debe asegurar en primer lugar que se tenga acceso como administrador (*root*) al sistema de archivos de los distintos dispositivos utilizados. Luego, mediante ADB (Android Debug Bridge) y Buildroot (herramienta que permite generar sistemas Linux embebidos), se

puede compilar el código. El mismo debe ser compilado para la arquitectura ARM. Cada fichero generado por este proceso debe ser copiado a cada dispositivo que forme parte de la red de procesamiento. En el archivo *hosts* de cada dispositivo se deberá almacenar la dirección IP y el nombre del host de cada dispositivo. [2]

Una vez realizada la configuración inicial, pueden crearse aplicaciones o scripts que inicien la ejecución del servidor y de los diversos clientes. La tarea fundamental será analizar todos los datos recopilados de los Molestadores (los mismos transfieren la información al teléfono/tablet) mediante conexión Bluetooth. Estos datos son enviados mediante los sockets al servidor, quien recopilará toda la información de los dispositivos recibida. Debido a la escalabilidad buscada en el proyecto, no es una decisión correcta almacenar la información de manera distribuida entre los equipos clientes que ejecutarán algoritmos de cálculo, sino que corresponde hacer uso de una base de datos central instalada en el servidor, y accedida para lectura y escritura por el mismo.

El servidor tendrá en todo momento una lista actualizada de dispositivos conectados, además de llevar nota de la información nueva que recibe. Dicha información estará formada por, entre otros valores:

- Hora de configuración de la alarma.
- Hora en la que la alarma sonará.
- Duración de la alarma programada desde que se configuró, hasta su disparo.
- Hora de desactivación de la alarma.
- Tiempo transcurrido entre el disparo de la alarma y su desactivación.
- Datos de ubicación para generar informes más detallados.

El servidor distribuirá la información y el algoritmo a los dispositivos Android que forman la red, teniendo en cuenta la utilización de CPU de cada uno, para balancear la carga y evitar cuellos de botella. Una vez que los clientes ejecutan lo solicitado, envían la información final hacia el servidor, quien la almacenará en una base de datos adicional, con informes finales con patrones de uso entre los usuarios de Molestador.

### **3 Explicación del algoritmo.**

La funcionalidad básica del algoritmo será detectar patrones irregulares en el sueño de los usuarios de Molestador. La característica fundamental que tendrá que analizar será la duración de las alarmas programadas, y la hora efectiva de desactivación. Uno de los datos enviados por parte del Molestador es la hora de configuración de la alarma, y es importante para entender si el conjunto de datos a analizar pertenece a un horario nocturno (donde se deben respetar las 8 horas de descanso), o bien a un horario diurno (donde puede estar analizándose una siesta, donde no se tiene en cuenta el mínimo de horas de sueño). Si bien existe un margen de error en los datos

recabados y en el algoritmo, hechos como programación seguida de alarmas, alarmas diurnas de poca duración, mucho tiempo transcurrido entre disparo y desactivación, horario tardío de configuración, permiten suponer que el descanso de la persona no fue satisfactorio. El algoritmo, en base a los datos enviados por el Molestador, y con la base de datos de conocimientos ya instalada en el servidor, permitirá analizar y estudiar patrones de sueño entre los usuarios. Dicha información podrá ser usada para realizar estadísticas sobre la calidad del descanso de las personas.

Los datos de ubicación serán útiles para detectar o realizar generalizaciones sobre determinadas regiones. Por lo tanto, en lugares con mayor acceso a servicios digitales o donde hayan grandes eventos disponibles, pueden verse patrones que son completamente opuestos a lugares en donde el contexto no sea el mismo.

Pseudocódigo del servidor:

```
/** Hilo principal **/  
inicializar archivo de configuración  
crear socket servidor  
conectar a la base de datos  
mientras no se cierre el servidor  
    ejecutar hilo de datos  
fin mientras  
cerrar archivos abiertos  
desconectar clientes  
desconectar servidor  
  
/** Hilo de conexión **/  
escuchar socket servidor  
mientras socket abierto  
    por cada cliente que busca conectarse  
        crear socket cliente  
        probar socket cliente  
        almacenar socket cliente en una lista  
    fin por  
fin mientras  
  
/** Hilo de datos **/  
por cada dato recibido  
    almacenar dato en base de datos  
    seleccionar un cliente conectado  
    si carga del cliente es baja  
        enviar datos  
        enviar algoritmo  
    fin si  
esperar resultados  
almacenar resultados
```

*fin por*

Pseudocódigo del cliente:

```
/** Hilo principal **/  
inicializar archivo de configuración  
crear socket  
conectarse con el servidor  
si conexión satisfactoria  
    iniciar hilo de datos  
caso contrario  
    cerrar archivo de configuración  
    fin de ejecución  
fin si  
mientras hilo de datos procese  
    dormir hilo temporalmente  
fin mientras  
cerrar archivo de configuración  
cerrar socket  
  
/** hilo de datos **/  
escuchar datos del servidor  
mientras servidor y cliente no sean cerrados  
    por cada dato recibido  
        si es dato del Molestador  
            enviar telemetría al servidor  
            enviar estado actual del dispositivo  
        si es dato del servidor  
            recibir datos para procesar  
            recibir algoritmo  
            procesar(datos, algoritmo)  
            enviar resultado final  
        fin si  
    fin por  
fin mientras
```

MPICH ofrece comandos que deberán ser utilizados tanto en el servidor como en el cliente para la conexión y para el intercambio de datos. *mpiexec* [3] es uno de ellos, y es el que inicializa el servidor, creando el archivo de configuración y distribuyéndolo a los clientes. El archivo de configuración posee la lista de clientes que forman la red de dispositivos. *smpd* [3] es otro comando que se puede utilizar para la inicialización del servidor, variando únicamente los parámetros enviados. La conexión a los sockets, el intercambio de información en los mismos, y la creación de hilos se mantienen utilizando el código de Android. *mpiexec* [3] permite enviar como parámetro el código MPI y la cantidad de procesos distribuidos deseados.

La comunicación se realiza de manera segura a través de ssh, y como se dijo anteriormente, es necesario tener acceso root al sistema de archivos. Cada dispositivo tendrá un par de claves públicas y privadas, permitiendo el cifrado RSA de los datos que se intercambian.

#### **4 Pruebas que pueden realizarse**

Para poder lograr un buen set de datos almacenados y perfeccionar el algoritmo utilizado, se necesita que una buena cantidad de Molestadores corran en paralelo recolectando y enviando datos al servidor principal. La información reunida tiene que ser lo más variada posible, con todo tipo de duración de alarmas, tanto en horarios nocturnos como diurnos. De esta manera, el algoritmo, en base a la información acotada que recibe, logra perfeccionar el nivel de patrones generados y estudiados.

La ubicación, tomada desde los dispositivos móviles, debe ser distribuida en distintos puntos, pero dentro de ellos, lo suficientemente densa como para poder realizar aseveraciones del sueño de las personas en áreas distintas.

#### **5 Conclusiones**

La investigación permitió dar a conocer un entorno de procesamiento distribuido y paralelizado en entornos Android. Tanto MPI como openMPI surgieron como candidatos naturales para realizar la tarea, pero debido a detalles de la arquitectura del sistema operativo de Google, su aplicación directa no es posible. MPICH surgió como una nueva alternativa que permite escalar y elevar el nivel de funcionalidades presentadas por el proyecto Molestador.

Sin dudas, la información recabada permitió conocer nuevas tecnologías aplicadas al área de la computación de alto rendimiento. Si bien el algoritmo presentado no conlleva una complejidad elevada, el volumen de datos alto puede hacer que este tipo de soluciones sea común en el día a día de las personas, y Molestador sería pionero en incluirlas.

#### **6 Referencias**

1. Yannes, Zachary, Tyson, Gary Scott, Wang, Zhi, Yuan, Xin.: Portable MPICH2 Clusters with Android Devices. Florida State University, College of Arts and Sciences, Department of Computer Science. Florida, United States. (2015)
2. Attia, Doha, ElKorany, Abeer Mohamed, Moussa, Ahmed.: High Performance Computing Over Parallel Mobile Systems. Department of Computer Science. Faculty of Computers and Information, Cairo University. Cairo, Egypt. (2016).

3. Amer Abdelhalim, Balaji Pavan, Bland Wesley, Gropp William, Guo Yanfei, Latham Rob, Lu Huiwei, Oden Lena, Peña Antonio, Raffenetti Ken, Seo Sangmin, Si Min, Thakur Rajeev, Zhang Junchao, Zhao Xin.: MPICH User's Guide. Mathematics and Computer Science Division. Argonne National Laboratory. (2017).