Implementacion de MPI en un invernadero Iot

Juan Valenzuela, Nicolas Fernandez, Mauro Nieva, Ezequiel Silvero

[1]Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina
valenzuelajuansantiago@gmail.com, ni.ko94@hotmail.com,
mauro_ramones_86@hotmail.com, ezequiel.silvero@hotmail.com

Resumen. Con la presente investigación vamos a tratar de demostrar de que forma podriamos aplicar un cluster MPIHC a nuestro proyecto.

Palabras claves: MPI, openMPI, MPICH, Android, Terrarium, Arduino.

1 Introducción

El proyecto Terrarium fue pensado para controlar por medio de Arduino un invernadero. Por intermedio de un sensor bluetooth se envian los datos de las mediciones de temperatura a la app mobile. A su vez la app tambien es un gateway ya que por medio de la funcionalidad modo inteligente se traen de un Api Rest los rangos de temperatura y humedad., en base a la ciudad y tipo de suelo.

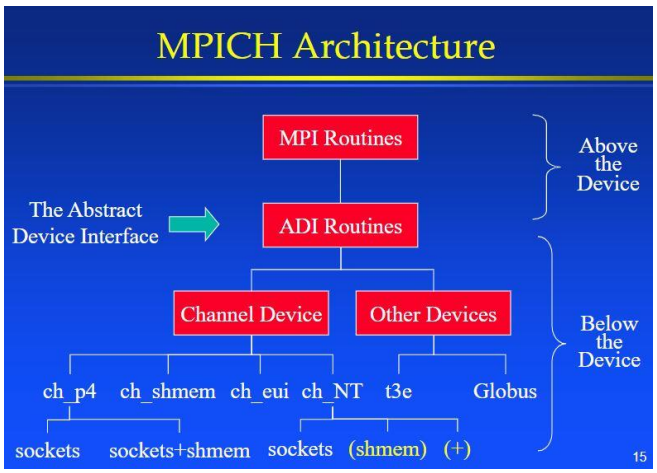
Con dichos rangos le permiten al sistema embedido tomar la decisión de si hay que regar o ventilar el ambiente.

Actualmento usando MPIHC proponemos darle una mejora a nuestro proyecto la posibilitar de montar un cluster entre distintos disposivos que poseen instalada la app mobile para tomar metricas globales sobre usos de datos , tiempo de ventilacion y riego. Con dichos tiempos y la ubicación geografica se podria calcular el costo promedio de la energia electrica consumida.

2 Desarrollo

MPICH distribuye el código en dos secciones:
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 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.

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. [3]

Una vez realizada la configuración inicial, pueden crearse aplicaciones o scripts que inicien la ejecución del servidor y de los diversos clientes.

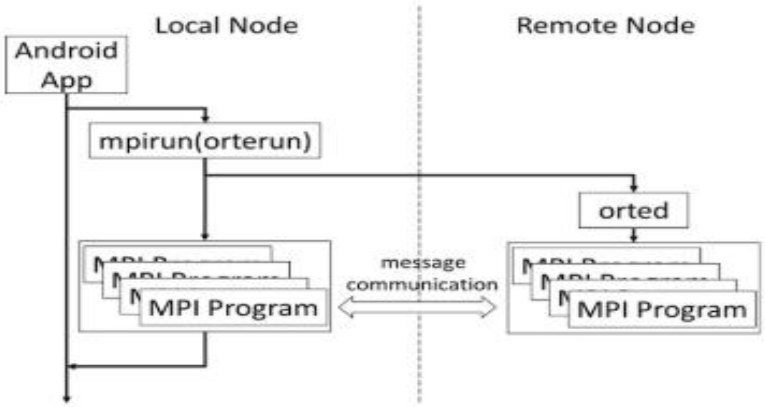


Fig. 3. MPI parallel processing on Android application

La tarea fundamental será analizar todos los datos recopilados de los invernaderos 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 hora de inicio y fin de riego, hora de inicio y fin ventilación, uso de datos.
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.

3 Explicación del algoritmo.

Una vez que el cliente solicite obtener datos sobre los tiempos totales de riego, ventilación y los costos va a hacerle una petición al servidor para obtener los mismos. El servidor al tener ya una base de datos con todos los datos propios de los clientes procederá a hacer las correspondientes operaciones y luego le enviará los datos requeridos.
Una vez que se finalice el hilo de datos en el cliente se cierra el socket y posteriormente el servidor lo elimina de la lista.

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
calcular total de datos usados
calcular consumo total ventilación y riego
calcular costo de energía
seleccionar un cliente conectado
si carga del cliente es baja
enviar datos
fin si
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
enviar estado actual del dispositivo
si es dato del servidor
recibir datos para procesar
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 [4] 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 [4] 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 [4] 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 se necesita que haya conectado una cantidad grande de clientes, corriendo en paralelo recolectando y enviando datos al servidor principal.

5 Conclusiones

Si bien se pudo demostrar que es viable la investigación, los procesos paralelos son iniciados por la máquina virtual de Android y hay una posibilidad que esto produzca overhead y esto ocasiona una baja performance a la hora de hacer procesamiento paralelo.

Como alternativa a futuro podríamos considerar el uso de OpenMP ya que tendríamos más capacidad de procesamiento y un poco más de paralelismo.

6 Referencias

1. Nissato M., Sugiyama H., Ootsu K., Ohkawa T., Yokota T. (2020) Realization and Preliminary Evaluation of MPI Runtime Environment on Android Cluster. In: Barolli L., Takizawa M., Xhafa F., Enokido T. (eds) Advanced Information Networking and Applications. AINA 2019. Advances in Intelligent Systems and Computing, vol 926. Springer, Cham
2. 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)
3. 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).