



**UNIVERSIDAD POLITÉCNICA SALESIANA**

**SEDE QUITO**

**CARRERA DE INGENIERÍA DE SISTEMAS**

**Procesamiento de archivos fuente de datos que provienen de las estaciones hidrometeorológicas automáticas.**

Trabajo de titulación previo a la obtención del

Título de: Ingeniero de sistemas

AUTOR: ADNAN ISAAC CÁCERES ZURITA

TUTORA: PAULINA ADRIANA MORILLO ALCÍVAR

Quito-Ecuador

2022

**CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE  
TITULACIÓN**

Yo, Adnan Isaac Cáceres Zurita, con documento de identificación N° 1722362603, manifiesto que: Soy autor y responsable del presente trabajo; y, autorizo a que sin fines de lucro la Universidad Politécnica Salesiana puede usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Quito, 14 de septiembre de 2022



.....  
Adnan Isaac Cáceres Zurita

1722362603

**CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE  
TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA**

Yo, Adnan Isaac Cáceres Zurita, con documento de identificación N° 1722362603, manifiesto mi voluntad y cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del Proyecto Técnico: “Procesamiento de archivos fuente de datos que provienen de las estaciones hidrometeorológicas automáticas”, mismo que ha sido desarrollado para optar por el título de: INGENIERO DE SISTEMAS, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribo este documento en el momento que hago la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Quito, 14 de septiembre de 2022

Atentamente,



.....

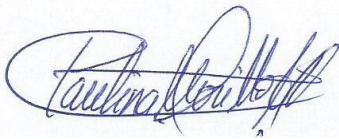
Adnan Isaac Cáceres Zurita

1722362603

**CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE  
TITULACIÓN**

Yo, Paulina Adriana Morillo Alcívar con documento de identificación N° 1715646574, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación PROCESAMIENTO DE ARCHIVOS FUENTE DE DATOS QUE PROVIENEN DE LAS ESTACIONES HIDROMETEOROLÓGICAS AUTOMÁTICAS, realizado por Adnan Isaac Cáceres Zurita con documento de identificación N° 1722362603, obteniendo como resultado final el trabajo de titulación bajo la opción proyecto técnico que cumple con todos los requisitos por la Universidad Politécnica Salesiana.

Quito, 14 de septiembre de 2022



Ing. Paulina Adriana Morillo Alcívar, MSc

1715646574

## **DEDICATORIA**

Dedico este proyecto de grado primero a Dios por haberme dirigido por el sendero correcto. A mis padres, Segundo Cáceres y Luz Zurita, por ese apoyo incondicional que me han brindado para cumplir uno de mis grandes sueños. A mis hermanos, Ishmael Cáceres, Carolina Cáceres y Cáterin Cáceres por su apoyo a lo largo de mi formación profesional. Al Ing. Darwin Rosero por guiarme dentro del INAMHI. A mi tutora Ing. Paulina Adriana Morillo Alcívar por guiarme con su conocimiento y apoyo en la culminación de mi proyecto.

**Adnan Isaac Cáceres Zurita.**

# ÍNDICE GENERAL

INTRODUCCIÓN .....	1
CAPÍTULO I .....	7
1 MARCO TEÓRICO .....	7
1.1 <i>Herramientas de Desarrollo del Programa</i> .....	7
1.1.1 <i>Visual Studio Code</i> .....	7
1.1.2 <i>Virtual Box</i> .....	7
1.1.3 <i>Pycharm Community</i> .....	7
1.1.4 <i>GitHub</i> .....	8
1.1.5 <i>Lenguajes de Programación.</i> .....	8
1.1.6 <i>Librerías de programación.</i> .....	8
1.1.7 <i>Base de Datos (BDD)</i> .....	10
1.1.8 <i>Protocolos de comunicación.</i> .....	12
1.2 <i>Big Data</i> .....	13
1.2.1 <i>Volumen de Datos</i> .....	13
1.2.2 <i>Velocidad en que se recibe los Datos</i> .....	13
1.2.3 <i>Variedad de Datos</i> .....	13
1.3 <i>Funcionamiento del Paralelismo.</i> .....	14
1.3.1 <i>Ventajas del Paralelismo.</i> .....	14
1.3.2 <i>Desventajas del Paralelismo</i> .....	14
1.3.3 <i>Scheduling.</i> .....	15
1.3.4 <i>Multithreading</i> .....	16
1.3.5 <i>Multiprocesamiento.</i> .....	17
1.3.6 <i>Ventajas de Multiprocesamiento sobre Multithreading</i> .....	17
1.4 <i>Estaciones Hidrometeorológicas</i> .....	18
1.4.1 <i>Estaciones Automáticas</i> .....	18
1.4.2 <i>Dataloggers.</i> .....	19
1.5 <i>Formato de encapsulación del mensaje</i> .....	20
1.5.1 <i>Nombre del archivo</i> .....	20
1.5.2 <i>Contenido del Archivo</i> .....	22

1.6	<i>Archivos no Procesados.</i>	26
1.7	<i>Umbrales para los archivos (. rep).</i>	27
1.8	<i>Metodologías Agiles</i>	28
1.8.1	<i>SCRUM.</i>	29
1.8.2	<i>Roles Scrum.</i>	29
<b>CAPÍTULO II</b>		31
2	<b>ANÁLISIS Y REQUERIMIENTOS.</b>	31
2.5	<i>Análisis de factibilidad.</i>	31
2.1.1	<i>Viabilidad Técnica.</i>	31
2.1.2	<i>Viabilidad Económica.</i>	34
2.1.3	<i>Viabilidad Operacional.</i>	36
2.6	<i>Análisis de Requerimientos.</i>	37
2.2.1	<i>Alcance.</i>	37
2.2.2	<i>Requerimientos técnicos.</i>	37
<b>CAPÍTULO III</b>		39
3	<b>DESARROLLO</b>	39
3.5	<i>Arquitectura</i>	39
3.6	<i>Desarrollo del programa en Phyton3.</i>	41
3.6.1	<i>Limpieza de los datos obtenidos del servidor FTP</i>	41
3.6.2	<i>Buscar y cargar los datos umbrales.</i>	42
<b>CAPÍTULO IV</b>		44
4	<b>PRUEBAS Y RESULTADOS</b>	44
4.1	<i>Comparación de rendimiento de librerías para procesamientos de datos en Python.</i>	44
4.2	<i>Módulos y ficheros de prueba.</i>	45
<b>CONCLUSIONES</b>		52
<b>RECOMENDACIONES</b>		53
<b>REFERENCIAS</b>		54

## ÍNDICE DE FIGURAS

<b>Figura 1</b> Recepción, procesamiento y envío de datos hacia la BDD. ....	4
<b>Figura 2</b> Funcionamiento del Paralelismo. ....	15
<b>Figura 3</b> Sincronización de los procesos paralelos en memoria. ....	16
<b>Figura 4</b> Estación Hidrometeorológica Automática. ....	20
<b>Figura 5</b> Formato del nombre del Archivo Fuente. ....	22
<b>Figura 6</b> Contenido del Archivo Fuente. ....	23
<b>Figura 7</b> Ejemplo del formato de los Archivos Fuente ....	24
<b>Figura 8</b> Estructura del Id Nemónico ....	25
<b>Figura 9</b> Carpeta de Archivos no procesados ....	26
<b>Figura 10</b> Estructura de la Matriz umbrales ....	27
<b>Figura 11</b> Arquitectura de los módulos del proyecto ....	39
<b>Figura 12</b> Diagrama de flujo ....	40
<b>Figura 13</b> Limpieza de los datos obtenidos del servidor FTP ....	42
<b>Figura 14</b> Buscar y cargar datos umbrales. ....	43
<b>Figura 15</b> Tiempo promedio Numpy vs Pandas. ....	44
<b>Figura 16</b> Tiempo de ejecución monoprocso. ....	46
<b>Figura 17</b> Consumo de CPU en monoprocso. ....	46
<b>Figura 18</b> Consumo de RAM en monoprocso. ....	47
<b>Figura 19</b> Tiempo de ejecución multiprocso. ....	48
<b>Figura 20</b> Consumo de CPU en multiprocso ....	49
<b>Figura 21</b> Consumo de RAM en multiprocso ....	49
<b>Figura 22</b> Tiempo de ejecución multiprocso vs monoprocso ....	50



## ÍNDICE DE TABLAS

<b>Tabla 1</b> Características de Postgres y MongoDB. ....	11
<b>Tabla 2</b> Ventajas de Multiprocesamiento sobre Multithreading .....	17
<b>Tabla 3</b> Códigos del tipo de medio de comunicación.....	21
<b>Tabla 4</b> Requisitos del Hardware.....	32
<b>Tabla 5</b> Requisitos del Software. ....	32
<b>Tabla 6</b> Costos del desarrollo del Programa.....	34
<b>Tabla 7</b> Costos de Hardware.....	35
<b>Tabla 8</b> Costos de Servicios.....	35
<b>Tabla 9</b> Costo de recursos.....	36
<b>Tabla 10</b> Requerimientos técnicos.....	38

## RESUMEN

Este proyecto tiene como finalidad realizar el procesamiento de archivos fuente de datos que provienen de las estaciones hidrometeorológicas automáticas del Instituto Nacional de Meteorología e Hidrología del Ecuador. Los archivos fuentes que transmiten las estaciones automáticas del INHAMI se reciben en el servidor cada minuto. Estos archivos contienen toda la información programada por los *datalogger* que incluyen en promedio 60 parámetros meteorológicos.

Por lo tanto, este trabajo realizó el desarrollo de un programa para detectar archivos con errores, en función de las reglas establecidas por la institución. Los principales errores que se consideraron son archivos vacíos, con otros formatos, etc. Después de descartar los archivos con errores, se procesan los datos y se los almacena en una DB. La metodología empleada para el desarrollo del proyecto fue SCRUM. De esta forma, se logró cumplir con los requerimientos solicitados por parte de los usuarios finales del programa. Las herramientas empleadas para el desarrollo del proyecto incluyen el uso de *software* libre como Python con sus librerías Pandas y Numpy, las cuales facilitaron el cálculo y procesamiento de datos. De igual forma, se usó PostgreSQL para la base de datos. También mediante el uso de multiproceso los resultados que se obtuvieron al trabajar con ello demuestran ser factibles para este proyecto. De acuerdo a los resultados se logró disminuir la frecuencia de llegada de los archivos fuente, aumentando el tiempo de recepción a cinco minutos, de esta forma, se evita la congestión del servidor. Además, se logró filtrar los archivos (.rep), descartando ficheros con errores. Esto facilita la depuración de la información antes de que sea almacenada en la base de datos.

Finalmente, se concluye que el programa implementado ejecuta con éxito las instrucciones definidas por el INAMHI.

## **ABSTRACT**

This project aims to process data source files that come from automatic hydrometeorological stations of the National Institute of Meteorology and Hydrology of Ecuador. The source files transmitted by INHAMI's automatic stations are received on the server every minute. These files contain all the programmed information by the dataloggers that include on average 60 meteorological parameters.

Therefore, this work carried out the development of a program to detect files with errors, based on the rules established by the institution. The main errors that were considered are empty files, with other formats, etc. After discarding the files with errors, the data is processed and stored in a DB. The methodology used for the development of the project was SCRUM. In this way, it was possible to meet the requirements requested by the end users of the program. The tools used for the development of the project include the use of free software such as Python with its Pandas and Numpy libraries, which facilitated the calculation and processing of data. Similarly, PostgreSQL was used for the database. Also through the use of multithreading the results that were obtained by working with it prove to be feasible for this project. According to the results, it was possible to reduce the frequency of arrival of the source files, increasing the reception time to five minutes, thus avoiding server congestion. In addition, it was possible to filter the (.rep) files, discarding files with errors. This makes it easy to clean the information before it is stored in the database.

Finally, it is concluded that the implemented program successfully executes the instructions defined by INAMHI

## **INTRODUCCIÓN**

### **ANTECEDENTES**

El Instituto Nacional de Meteorología e Hidrología (INAMHI) es una entidad pública cuyo objetivo es generar y difundir información hidrometeorológica. Que, por medio de la ciencia y la tecnología actual, tiene la posibilidad de vigilar y predecir el comportamiento de la atmósfera y las aguas interiores, para el pronóstico diario del tiempo, predicciones y avisos de fenómenos meteorológicos e hidrológicos extremos.

Esta institución, también contribuye al esfuerzo internacional del control del tiempo y el clima, mediante el intercambio de información con otros países, ya que tiene representación nacional e internacional y es miembro de la Organización Meteorológica Mundial (OMM).

Una parte fundamental del trabajo de esta entidad consiste en el procesamiento de los datos hidrometeorológicos que se transmiten desde estaciones automáticas, a través de archivos fuente. Estas estaciones se encuentran en diferentes partes del Ecuador y miden variables como, presión atmosférica, radiación solar, velocidad del viento, turbiedad del agua, PH del agua, etc. Los archivos fuentes llegan en formatos comprimidos estándar y se transmiten por medios de comunicación inalámbricos, lo que ocasiona errores.

## **PROBLEMA DE ESTUDIO**

Actualmente, el INAMHI cuenta con estaciones automáticas que transmiten datos meteorológicos, usando la tecnología de comunicación GPRS (Servicio General de Paquetes vía Radio) y la conexión con los satélites GOES (*Geostationary Operational Environmental Satellite*). Estos satélites se encargan de controlar el clima en los Estados Unidos y sus alrededores, también envían datos hidrometeorológicos a diferentes países. En el caso del INAMHI, estos archivos se reciben en los servidores de la institución cada minuto, en formato comprimido F10 (Estandarizado) de extensión (. rep). Este formato de compresión provoca errores que no se pueden detectar hasta que llegan al servidor. Por consiguiente, se dificulta el procesamiento de los datos y su almacenamiento. Para mitigar este problema se busca etiquetar los valores que no cumplen determinadas reglas, para eliminarlos antes de guardarlos en la base de datos. Las reglas están normadas en la "Guía del Sistema Mundial de Observación" de la OMM Número 448 (Mundial, Guía del Sistema Mundial de Observación, 2010).

Este trabajo se enfoca en el procesamiento de la información una vez que el archivo se encuentra en formato de texto en los servidores del INAMHI, es ahí donde se plantea utilizar *scripts* para la verificación de los archivos, antes de que la información se almacene en la base de datos (PostgreSQL).

## **JUSTIFICACIÓN**

Los datos provenientes de las estaciones hidrometeorológicas automáticas son de suma importancia, debido a que, a través de ellos se calculan variables como la temperatura del aire, velocidad del viento, turbiedad del agua, radiación solar global, presión atmosférica, etc. Estas

variables son necesarias para implementar modelos matemáticos que pronostiquen el tiempo y el clima.

Hoy en día, el INAMHI recibe los datos hidrometeorológicos a través de archivos fuente en formato. rep, que en algunas ocasiones presentan errores o carecen de información. Además, debido a la frecuencia de llegada de estos ficheros, suelen haber congestiones de procesamiento en el servidor. Por lo tanto, es indispensable contar con *scripts* que verifiquen el contenido de estos archivos para evitar errores posteriores en el almacenamiento de los datos o consumo de recursos computacionales innecesarios.

La elaboración de este trabajo busca desarrollar un *script* que incorpore el proceso de verificación de los archivos fuente, para eliminar aquellos que no cumplen con las normas establecidas por el INAMHI, y de esta forma, disminuir los errores en los datos y aumentar la frecuencia de recepción de los archivos.

### **GRUPO OBJETIVO (Beneficiarios)**

El grupo que se beneficiaría al realizar este proyecto está conformado por los funcionarios del Departamento de información del INHAMI, ya que podrán tener una herramienta que aumente la calidad de los archivos fuente receptados y a su vez la calidad de los modelos de predicción.

### **OBJETIVOS**

#### **Objetivo general:**

- Realizar el procesamiento de archivos fuente de datos que provienen de las estaciones hidrometeorológicas automáticas.

#### **Objetivos específicos:**

- Estudiar el formato y estructura de los datos formato F10, para su posterior procesamiento.
- Implementar *scripts* que reduzcan el tiempo de procesamiento de datos enviados por las estaciones hidrometeorológicas automáticas.
- Implementar la programación paralela para gestionar grandes volúmenes de información (*BIG DATA*).

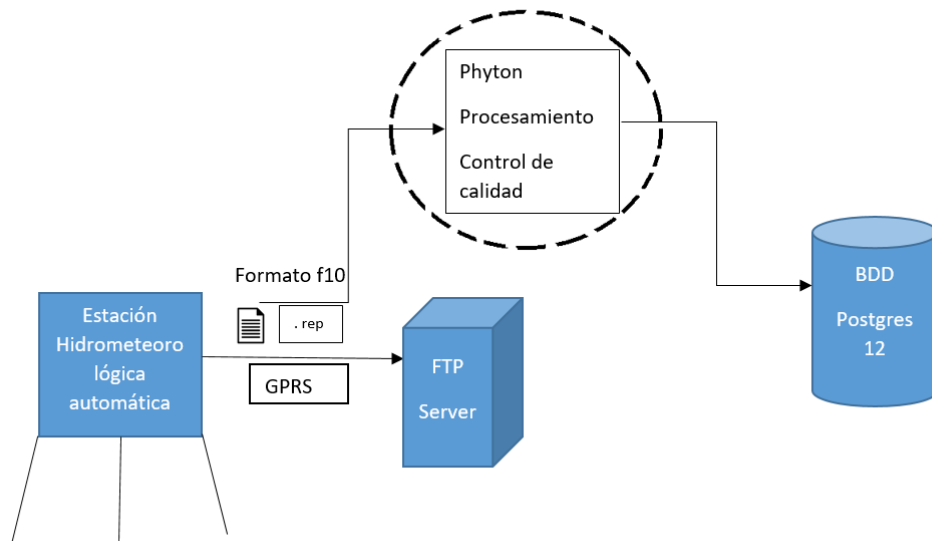
## METODOLOGÍA

Para la elaboración de este proyecto se utilizará la metodología ágil SCRUM, ya que es necesario seguir un conjunto de buenas prácticas en el desarrollo de la aplicación. Además, esta metodología permite el trabajo colaborativo y en equipo y la mejora continua del producto.

El proceso que tienen estos archivos se muestra en la **Figura 1** la cual empieza con la estación hidrometeorológica automática, esta estación es la que genera los archivos .rep, mediante GPRS estos archivos se dirigen hacia los servidores FTP del INAMHI, para luego realizar el procesamiento y control de calidad de los mismos, en esa sección es donde nos enfocaremos para finalmente enviar los datos de forma automática a la Base de Datos.

### **Figura 1**

*Recepción, procesamiento y envío de datos hacia la BDD.*



*Nota.* Recepción de datos, verificación del formato, procesamiento de archivos fuente, control de calidad y envío de datos hacia la BDD. Elaborado por Adnan Cáceres.

- **RECEPCIÓN DE DATOS:** Los datos son enviados por medio de una trama hacia las estaciones hidrometeorológicas automáticas, estos datos se transmiten mediante GPRS hacia los servidores FTP (código GFTP).
- **VERIFICACION DEL FORMATO:** Los datos deben de encontrarse en el formato f10(Formato de archivo estándar utilizado por el INAMHI). Estos archivos se envían con terminación (. rep).
- **PROCESAMIENTO DE ARCHIVOS FUENTE:** Una vez se reciba los archivos (. rep) con el formato f10 se realizará el procesamiento de los mismos.
- **CONTROL DE CALIDAD DE ARCHIVOS FUENTE:** Con los datos obtenidos se debe tener un control de calidad es analizar los datos para que estos no tengan valores incorrectos tales como temperaturas excesivamente altas o bajas, velocidad del viento con valores negativos, etc. Para controlar este tipo de errores se maneja mediante umbral.



- ENVIO DE DATOS HACIA LA BASE DE DATOS: Al finalizar el procesamiento, análisis y control de calidad de los datos, se procede a enviar hacia la base de datos realizada en Postgres 12.

Para la elaboración general de este proyecto se utilizan instrumentos de investigación para realizar el levantamiento de información, también se utiliza investigación de campo, bibliográfica y documental, para ello se utilizan técnicas de instrumentos como reuniones, entrevistas, etc.

### **Roles SCRUM.**

- Product Owner: Ing. Darwin Rosero
- Scrum Master: Msc. Paulina Morillo.
- Development team: Ing. Darwin Rosero, Adnan Cáceres
- Usuarios: Ing. Darwin Rosero.

# CAPÍTULO I

## 1 MARCO TEÓRICO

### 1.1 *Herramientas de Desarrollo del Programa*

Los programas informáticos que son utilizados para la creación, depuración, mantenimiento, dar soluciones o encontrar errores, incluso dar apoyo a programas y aplicaciones se les denomina como herramientas de programación o *software*.

Mediante el uso de programas que son fáciles, se las puede combinar para la realización de una tarea. La facultad para el uso de diversas herramientas de forma eficiente y sin la presencia de errores o inconvenientes se denomina como una cualidad de un buen ingeniero de software (EUROINNOVA).

**1.1.1 *Visual Studio Code*.** Es un editor de código fuente liviano y a su vez potente, el cual se lo ejecuta desde el escritorio, lo podemos encontrar disponible para diversos sistemas operativos tales como Windows, Linux, etc. Este editor tiene soporte para JavaScript, TypeScript y Node.js y posee una gran variedad de extensiones para diferentes lenguajes (como C++, C#, Java, Python, PHP) (OSWALDO, 2021).

**1.1.2 *Virtual Box*.** Es un *software* de virtualización, o monitor de máquinas virtuales. Se utiliza para crear máquinas dentro de un mismo ordenador de manera virtual, es decir, se puede tener varias computadoras con diferentes S.O en un solo ordenador anfitrión (Roberto Solé, 2020).

**1.1.3 *Pycharm Community*.** Ayuda con el reconocimiento del código, muestra donde se encuentran errores al momento de ir escribiendo el mismo, también permite

realizar arreglos de manera rápida, así como refactorización de código de una forma automática (JetBrains, 2022).

**1.1.4 *GitHub*.** Es una Herramienta con servicio en la nube que sirve para alojar proyectos de desarrollo de *software* en colaboración con otras personas, esta herramienta permite tener un control de versiones en el cual se pueden hacer cambios y dar un seguimiento del avance de los proyectos, de igual forma, se puede compartir el proyecto con las personas que lo requieran (Gustavo B., 2021).

**1.1.5 *Lenguajes de Programación*.** Un lenguaje de programación es definido como un grupo de normas o reglas que permiten la comunicación con las computadoras para el desarrollo de programas, aplicaciones, páginas webs, scripts u otros grupos de procesos (Marco A. Peña Basurto, 2001). A continuación, se describen los lenguajes utilizados en este proyecto.

**Python3:** es un lenguaje de programación de alto nivel creado en los años noventa por Guido Van Rossum, tras el pasar de los años ha tenido un mayor crecimiento llegando a ser uno de los lenguajes más usados en la actualidad, ya que puede ser utilizado en diferentes plataformas y S.O como Linux, Windows, etc. (Montoro, 2012).

**JAVA:** Java es un lenguaje de programación de un alto rango, orientado a objetos, con el cual podemos desarrollar programas tanto habituales como para Internet (Sierra, 2015).

**1.1.6 *Librerías de programación*.** Una librería es un grupo de ficheros los cuales son utilizados para la creación de *software*. Estas librerías acostumbran estar formadas

por datos y código, la finalidad de estas es ser utilizadas por otros programas. Es decir, es un fichero que puede ser importado (devCamp, 2020).

Las librerías más comunes para cálculo numérico se describen a continuación:

**1.1.6.1 Numpy.** Es una librería que incluye diferentes funciones básicas, Numpy representa matrices multidimensionales, según (Ivet Challenger, Diaz, & Becerra, 2014) “Numpy ha dado acceso a un gran número de funciones científicas que compiten en tamaño y velocidad con Matlab”.

**1.1.6.2 Pandas.** Es un envoltorio del lenguaje Python que brinda formas de datos veloces, manejable y expresivas diseñadas para trabajar con datos almacenados en hojas de cálculo. (the pandas development team, 2008).

**1.1.6.3 Psycopg2.** Es el adaptador de base de datos PostgreSQL más conocido para el lenguaje de programación Python. Psycopg 2 se implementa principalmente en C como un envoltorio alrededor de libpq, lo que lo hace eficiente y seguro. Tiene cursores del lado del cliente y del lado del servidor, comunicación asíncrona y notificaciones, soporte COPY (Federico Di Gregorio, 2021).

**1.1.6.4 Ftplib.** La clase FTP lleva a cabo un cliente del protocolo FTP. Puede ser usado para escribir programas de Python para realizar varios trabajos FTP automatizados (Python Software Foundation).

**1.1.6.5 Multiprocessing.** Es un paquete que permite generar con una API similar al módulo de subprocesamiento. El paquete de multiprocesamiento proporciona simultaneidad local y remota, omitiendo el bloqueo del intérprete global mediante el uso de subprocesos (Python Software Foundation).

**1.1.6.6 Import time.** El método *time* () devuelve el tiempo como un número de punto flotante expresado en segundos desde la época, en UTC. Aunque la hora siempre se devuelve como un número flotante, no todos los sistemas proporcionan la hora con una precisión superior a 1 segundo. Si bien esta función normalmente devuelve valores no decrecientes, puede devolver un valor más bajo que una llamada anterior si el reloj del sistema se ha atrasado entre las dos llamadas (tutorialspoint).

**1.1.7 Base de Datos (BDD).** Una BDD es una compilación organizada de información o datos estructurados, los cuales se encuentran guardados de forma electrónica en un sistema informático. Normalmente, los usuarios pueden hacer diferentes operaciones a las bases de datos tales como eliminar, insertar, actualizar, modificar, etc., (Date, 2001).

Para el presente proyecto se migrará de MongoDB hacia PostgreSQL.

**MongoDB:** Es una base de documentos no SQL orientada a documentos de código abierto escrito en C++, creada por Dwight Merriman, Eliot Horowitz y Kevin Ryan<sup>1</sup> en el año 2007, caracterizada por su balance entre rendimiento y funcionalidad, debido a su sistema de consulta de contenidos (Sarasa, 2016).

**PostgreSQL:** fue diseñada como una base de datos orientada a objetos. PostgreSQL utiliza un patrón cliente-servidor y usa multiprocesos en sucesión de multihilos para confirmar la consistencia del sistema, además, es un gestor de base de datos *open source* y disponible sin ningún tipo de costo (M, L, & F, 2017).

El uso de Postgres se aplica cuando los datos requieren de un alto nivel de seguridad, por lo tanto, este proyecto usa la base de datos Postgres por encima de MongoDB, debido a las características que se muestran en la **Tabla 41**.

**Tabla 1**

*Características de Postgres y MongoDB.*

Postgres	MongoDB
Postgres utiliza SQL para acceder, manipular y definir datos en la base de datos. Tiene su propio SQL llamado PL/pgSQL	BSON es uno de los aspectos más importantes de MongoDB, ya que agiliza las consultas.
Se utiliza en varios sistemas bancarios, de fabricación e inteligencia empresarial.	MongoDB carece de esquema, se puede almacenar datos en forma de documento JSON, lo que proporciona flexibilidad refiriéndose a la modificación del esquema.
Es ideal para los flujos de trabajo transaccionales. Tiene incorporado mecanismos de seguridad. Las transacciones de Postgres se rigen por los principios ACID	Al ser una base de datos NoSQL, no suele cumplir con propiedades ACID. Por esta razón, equipos y desarrolladores no confían en estas bases de datos.

*Nota. Características de Postgres y MongoDB .Fuente: (elFaroStudio, s.f.).*

**1.1.8 *Protocolos de comunicación.*** Para la publicación de la información se utilizará el protocolo de transferencia de archivos FTP y el protocolo de control de transmisión TCP. Una breve descripción de ambos protocolos se detalla a continuación:

**FTP (File Transfer Protocol)**

FTP es un registro de red que sirve para enviar archivos que se encuentren conectados a una red TCP (Transmission Control Protocol), este protocolo está fundamentado en la arquitectura cliente-servidor. Esta arquitectura nos dice que desde un cliente podemos conectarnos a un servidor para poder enviar o descargar archivos o fichero sin importar el tipo de S.O que se utilice (J. Postel, 1985).

**TCP (Transmission Control Protocol, Protocolo de Control de Transmisión)**

Es un protocolo que permite crear conexiones entre equipos de comunicación a través de las cuales puede enviarse una gran cantidad de datos. El protocolo nos da la seguridad de que al momento de enviar datos estos llegaran sin ningún tipo de error y en el orden en el que fue enviado. También brinda un mecanismo para identificar diferentes tipos de aplicaciones que se encuentren en un mismo ordenador, a través del concepto de puerto (Postel, 1981).

## **1.2 Big Data**

Se denomina *Big Data* a aquellos datos que tienen las tres Vs (Volumen, Velocidad, Variedad). En breves palabras, los grandes datos son un grupo compuesto por datos complejos y de gran tamaño. Estos grupos de datos tienen un volumen tan grande que un programa de procesamiento de datos convencional se le es imposible o difícil de administrar (Oracle).

### **1.2.1 Volumen de Datos.**

Con *Big Data*, se realiza el procesamiento de altos volúmenes de datos que no son estructurados de baja consistencia. Los valores de estos datos no necesariamente deben ser conocidos, como ejemplo podemos tomar las fuentes de datos de diferentes redes sociales como Facebook, Instagram entre otras, las series de clics que se da en una página web. Para ciertas entidades organizacionales, esto podría ser varios terabytes de datos, o pueden ser varios de *petabytes* (Oracle)

### **1.2.2 Velocidad en que se recibe los Datos.**

Se refiere a la velocidad en la que los datos son recibidos y también se actúa sobre ellos. Normalmente, la velocidad con la que transmite los flujos de datos es transmitido de manera directa a la memoria y no se escriben en disco (SMART LIFE, s.f.).

### **1.2.3 Variedad de Datos.**

Se refiere a la cantidad de diferentes tipos de datos que existen, anteriormente los datos se encontraban estructurados por ello se los podía poner en BDD relacionales. Con el surgir de los Grandes Datos, los tipos de datos no estructurados y semiestructurados, como texto, audio, etc., necesitan un



procesamiento previo para obtener sentido y admitir metadatos (SMART LIFE, s.f.).

### ***1.3 Funcionamiento del Paralelismo.***

Con el paralelismo podemos realizar varias operaciones o cálculos de forma simultánea. Es decir, se puede realizar la ejecución de múltiples tareas al mismo tiempo, tomando en cuenta el principio de reducir los problemas grandes o complejos en varios problemas pequeños, para finalmente ser solucionados de forma paralela (Bernal, Albarracín, Gaona, & Nieto, s.f.).

#### ***1.3.1 Ventajas del Paralelismo.***

- Ayuda a solventar problemas que existían en el uso de un solo CPU.
- Reduce el tiempo de ejecución de los procesos en la CPU.
- Mejoramiento de Estructuras de mayor orden y complejidad.
- Acelera la ejecución de códigos dentro del entorno de gestión de CPU
- Permite resolver una mayor cantidad de procesos al mismo tiempo.
- Lograr resultados en menores tiempos de ejecución.
- Se pueden ejecutar varios procesos en paralelo(simultaneo).
- Las tareas pueden ser resueltas en diferentes partes.

#### ***1.3.2 Desventajas del Paralelismo.***

- Consumo excesivo de recursos de hardware de procesamiento.
- Complejidad en el desarrollo de programas.
- Perdida de datos al no sincronizarse correctamente.
- Demoras producidas por la comunicación entre tareas.

- Costos sumamente excesivos para la producción y el mantenimiento.

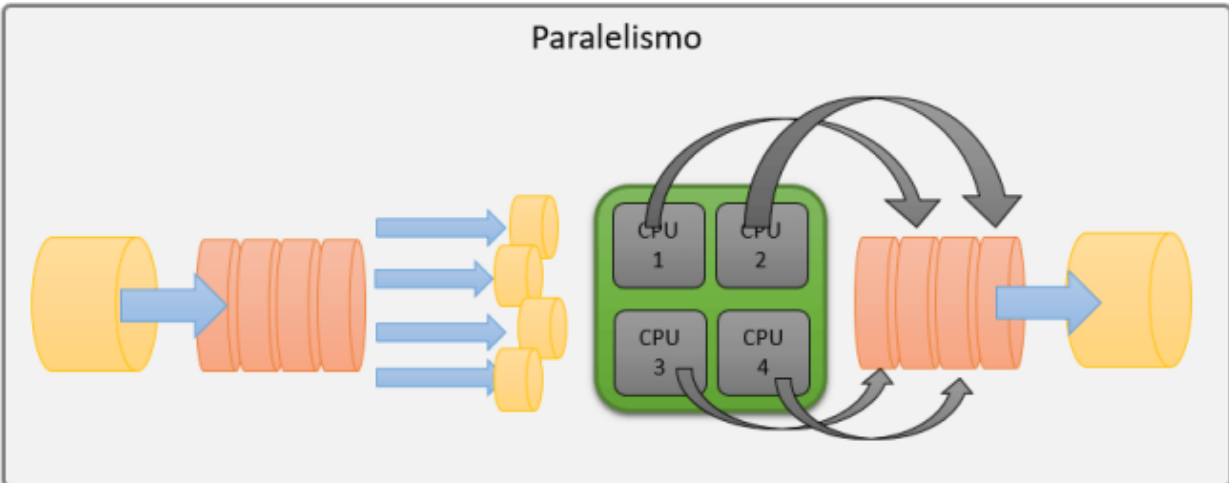
### **1.3.3 Scheduling.**

Es el transcurso en las cuales se asigna diferentes tareas a los procesos o hilos para darles una orden, para que puedan ser ejecutadas, el mismo que podemos especificar en el código que estemos desarrollando al momento que se esté compilando o de forma dinámica en el tiempo que se ejecute. El proceso de *scheduling* debe ser dependiente entre una tarea y otra, ya que, aunque algunas tareas son independientes, otras dependen de los datos de otros procesos (Bernal, Albarracín, Gaona, & Nieto, s.f.).

Al trabajar con gran cantidad de datos el paralelismo conlleva el uso de dos o más (cpus) como se observa en la **Figura 2**.

## **Figura 2**

*Funcionamiento del Paralelismo.*



*Nota. Ejemplo de ejecución en paralelo. Fuente: (Bernal, Albarracín, Gaona, & Nieto, s.f.).*

#### **1.3.4 Multithreading.**

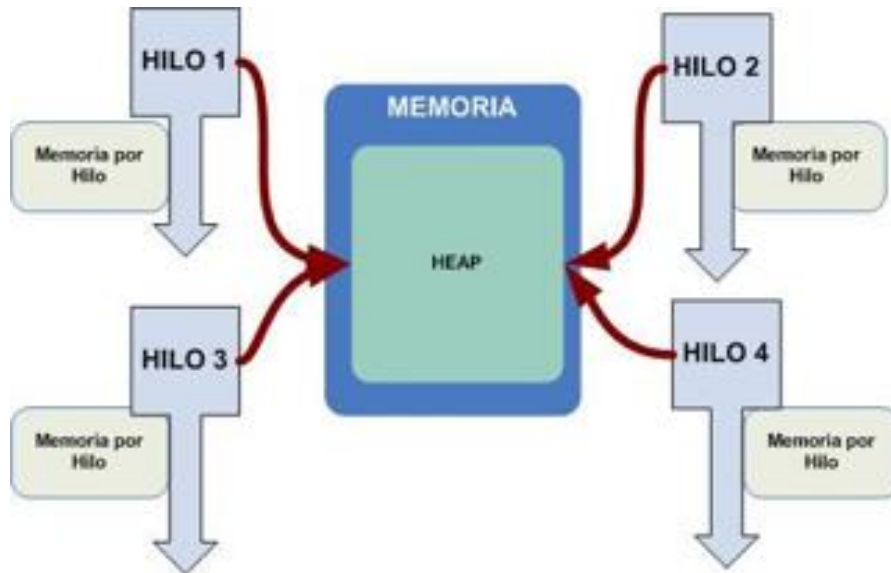
Es una técnica de programación que asigna múltiples segmentos de código a un único proceso. Estos segmentos de código, también denominados hilos, se ejecutan de forma concurrente y paralela entre sí. Los hilos están distribuidos en el mismo espacio de memoria dentro de un proceso principal.

Los programas que se ejecuten en paralelo deben combinar el uso de procesos e hilos, para que su ejecución sea correcta sin ningún tipo de inconveniente. Los procesos de coordinación y sincronización en la programación paralela se encuentran asociados fuertemente a la manera en que los procesos o hilos intercambian información, esto depende fundamentalmente del *hardware* y su organización (Bernal, Albarracín, Gaona, & Nieto, s.f.), como se muestra en la

**Figura 3.**

### **Figura 3**

*Sincronización de los procesos paralelos en memoria.*



*Nota. Se muestra como está organizada la memoria para tener una coordinación de procesos e hilos. Fuente: (Bernal, Albarracín, Gaona, & Nieto, s.f.).*

### 1.3.5 Multiprocesamiento.

“El multiprocesamiento se refiere a un sistema que tiene más de dos unidades centrales de procesamiento. Cada CPU adicional que se añade a un sistema aumenta su velocidad, potencia y memoria. Esto permite a los usuarios ejecutar varios procesos simultáneamente” (historiadelaprensa, s.f.).

### 1.3.6 Ventajas de Multiprocesamiento sobre Multithreading.

**Tabla 2**

*Ventajas de Multiprocesamiento sobre Multithreading*

Multiprocesamiento	Multithreading
La codificación que utiliza es fácil de entender	El código que utiliza puede ser difícil de entender
Los procesos pueden interrumpirse	No pueden ser interrumpidos
Completa las tareas más rápidamente y analiza grandes cantidades de datos.	La sobrecarga de diferentes hilos puede ser muy costosa para las tareas básicas.

*Nota. Ventajas que tiene el Multiprocesamiento sobre Multithreading. Elaborado por Adnan Cáceres a través de (historiadelaempresa, s.f.).*

#### **1.4 Estaciones Hidrometeorológicas.**

Para la implementación e instalación de las estaciones hidrometeorológicas es necesario basarse en las normas de la Organización Mundial-OMM las cuales dictan los protocolos para la toma y registros de datos que se debe tomar en cuenta. (Mundial, Guía de Instrumentos y Métodos de Observación Meteorológicos, 2010).

**1.4.1 Estaciones Automáticas.** Basándose en las normas de la OMM, las estaciones automáticas generalmente constan de:

- Múltiples sensores alojados en la parte alta media, baja y alrededor de las torres meteorológicas. Dichos sensores deben estar protegidos de los agentes externos ambientales, su sistema central debe estar conectado mediante cables rígidos de fibra óptica o mediante ondas de radio. (Hidrología, 2014).
- Es necesario configurar el SCP (Sistema central de procesamiento) de manera que se pueda recoger los datos en un formato legible y simple de procesar para un ordenador, este debe ser compatible con sistemas basados en microprocesadores en los cuales se pueden ejecutar algoritmos específicos para la recolección temporal de datos hidrometeorológicos distantes (Hidrología, 2014).
- Se debe suministrar una fuente de poder estabilizada y certificada que provea energía a las diversas partes de la torre, también es necesario incluir un reloj configurado con la zona horaria de la región a su vez es necesario implementar un equipo de escaneo de las diferentes partes de la estación para verificar su

estado y detectar con mayor rapidez los fallos ocasionados con el deterioro del tiempo (Hidrología, 2014)

El uso de las estaciones automáticas es diverso, como los siguientes:

- Facilita la obtención de datos de lugares de difícil acceso.
- Si el personal se encuentra fuera de los horarios de trabajo, facilita la observación de las estaciones.
- Aumentar la seguridad de datos, normalizar métodos y horarios en todas las estaciones de red.
- Si se encuartan estaciones que tengan mucho personal se puede menorar los gastos quitando esas estaciones.
- Realizar la instalación de sensores en lugares que sean favorables desde un punto de vista meteorológico.

**1.4.2 Dataloggers.** Es un mecanismo electrónico encargado de registrar datos que estén relacionados por su ubicación mediante el uso de sensores e instrumentos propios o ya sean conectados de forma externa. La mayoría se encuentran fundamentados en microcontroladores. Generalmente son diminutos pueden ser a baterías, portables y equipados con un microprocesador, tienen memoria interna y sensores (Final Test, s.f.).

En las estaciones automáticas del INAMHI los sistemas centrales de procesamiento se denominan *dataloggers*. Actualmente el INAMHI tiene diferentes tipos de *dataloggers* (diferentes fabricantes), como, por ejemplo: Vaisala, Logotronic, Campbell, etc. (Hidrología, 2014) como se observa en la **Figura 4**.

## Figura 4

*Estación Hidrometeorológica Automática.*



*Nota. Estación automática para Hidroagoyan en Baños: el sensor de duración de radiación de LSI LASTEM.*

*Fuente: (Provierto, 2014).*

## 1.5 Formato de encapsulación del mensaje

### 1.5.1 Nombre del archivo

El archivo debe ser nombrado con la siguiente información:

- Código de la estación. Corresponde al código del INAMHI que tiene asignada la estación (INAMHI, 2013).
- Identificador *Datalogger*. Este campo identifica al *datalogger* instalado en la estación. A nivel de base de datos este campo será relacionado con su respectivo número de serie y demás información (INAMHI, 2013).

- Código de tipo de formato. Cada tipo de *datalogger* viene con su propio *software* de administración y configuración. Hablando exclusivamente de la parte de configuración, se puede decir que cada *datalogger* propone diferentes características para crear los archivos de texto, los mismos que serán utilizados por las personas interesadas en la información (INAMHI, 2013).
- Tipo de medio. Este campo determina el medio por el cual se obtuvo el archivo, los cuales se encuentran en la **Tabla 3**.

**Tabla 3**

*Códigos del tipo del medio de comunicación.*

Código	Descripción
GPFT	Cuando el archivo es transmitido a través GPRS hacia el servidor FTP
STGS	Cuando el archivo es transmitido a través del satélite GOES
ETFT	Cuando el archivo es transmitido por Ethernet hacia el servidor FTP

*Nota. Se muestra los códigos que se utilizan para ver por qué medio se transmite los archivos. Elaborado por: Adnan Cáceres a través (INAMHI, 2013).*

- Tipo de información. Este campo determina si es información en tiempo real o en tiempo diferido por lo cual se usa R si es en tiempo real y D si es en tiempo diferido.

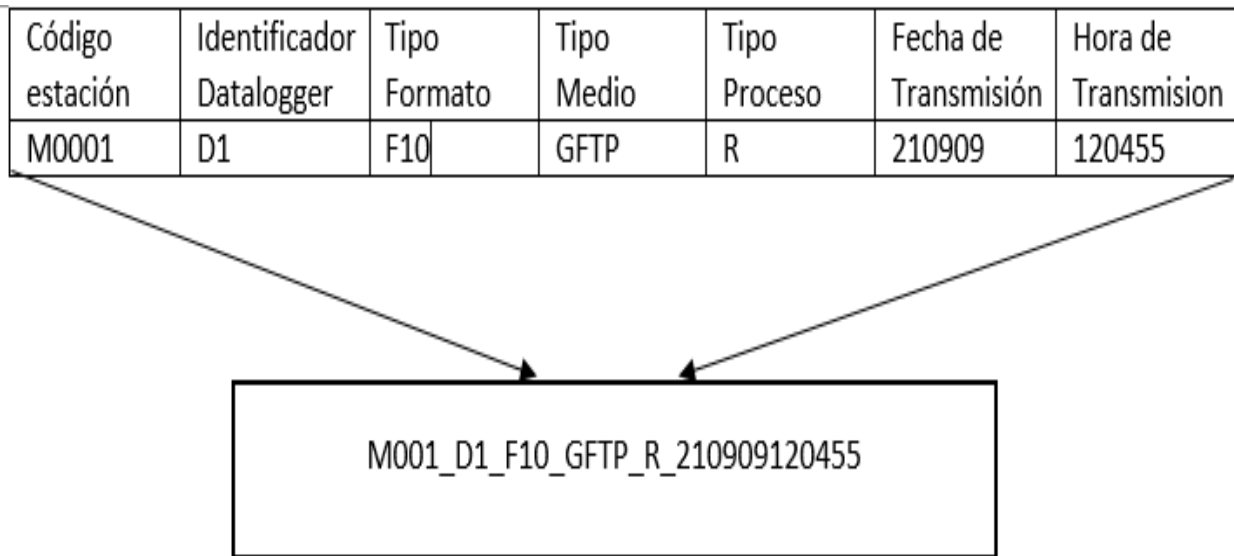


- Fecha en la que se transmitió el archivo. Este campo determina la fecha en la que se transmitió el archivo en formato YYMMDD (INAMHI, 2013).
- Hora en la que se transmitió el archivo. Este campo determina la hora en la que se transmitió el archivo en formato HHMMSS (INAMHI, 2013).
- Extensión del archivo. La extensión del archivo deberá ser .csv en todos los casos en los que la extensión pueda ser configurable (INAMHI, 2013).

La **Figura 5** muestra el formato que debe llevar el nombre de los archivos.

**Figura 5**

*Formato del nombre del Archivo Fuente.*



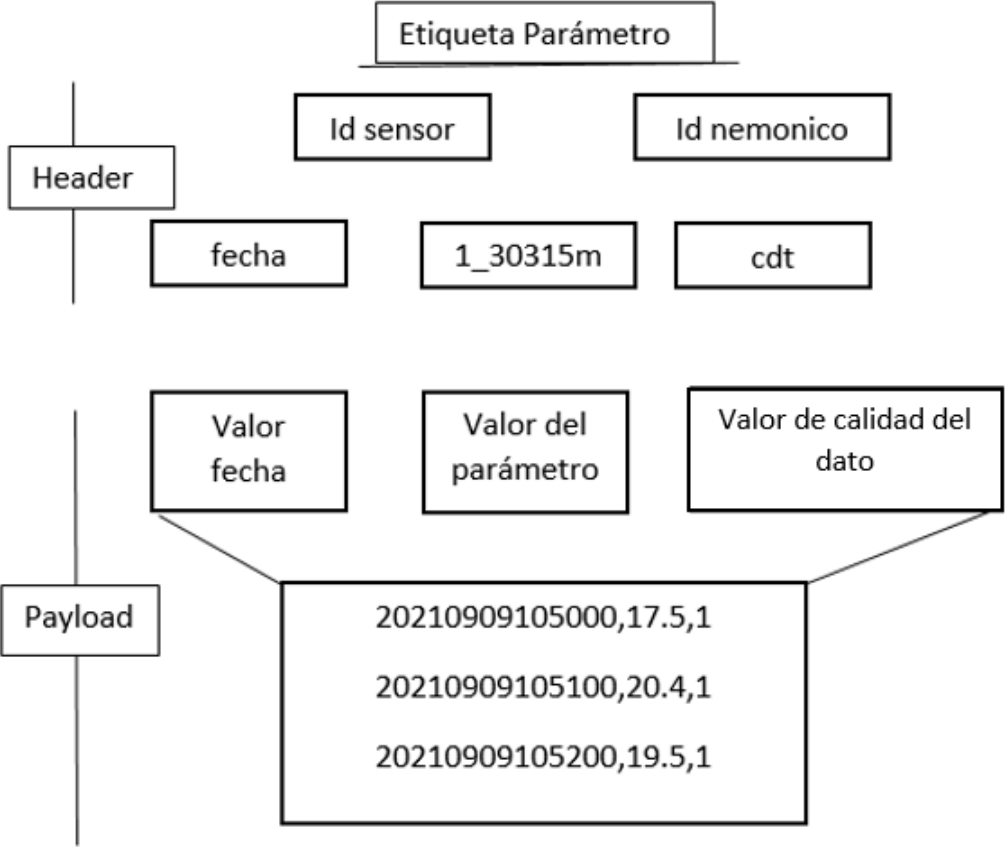
*Nota.* En esta figura se muestra el formato establecido por el INAMHI que debe llevar el nombre del archivo.  
Elaborado por Adnan Cáceres a través (INAMHI, 2013).

### 1.5.2 Contenido del Archivo

En la **Figura 6** se muestra el contenido del archivo, es decir, el *Header* el cual contiene la etiqueta del parámetro, el id del sensor, el id nemónico, la fecha y *Payload*, que indica los valores de los parámetros indicados en la primera fila.

**Figura 6**

*Contenido del Archivo Fuente.*



*Nota. Se muestra el formato que debe tener el archivo establecido por el INAMHI. Elaborado por Adnan Cáceres a través (INAMHI, 2013).*

En la **Figura 7** se muestra un ejemplo de cómo llegan los archivos, sus cabeceras empezando por la fecha, la siguiente columna es la cabecera, en la cual se encuentra el código del parámetro hidrometeorológico por ejemplo el recorrido del viento compuesto por el id del sensor y el id nemónico los mismos que se unen por (\_), el id nemónico se puede observar en la **Figura 8**.

### Figura 7

*Ejemplo del formato de los Archivos Fuente.*

fecha	11_544161m	cdt	1_29341m	cdt	1_29311m	cdt	1_29321m	cdt	1_293321m	cdt	1_9141m	cdt	1_9111m	cdt
2.02109E+13		0 55	11.7	0	12.37	0	11.46	0	0.293	0	95	0	96	0
2.02109E+13		0 55	11.63	0	11.94	0	11.46	0	0.187	0	95	0	96	0
2.02109E+13		0 55	11.75	0	12.36	0	11.46	0	0.272	0	95	0	96	0
2.02109E+13		0 55	11.76	0	12.25	0	11.46	0	0.281	0	95	0	95	0
2.02109E+13		0 55	11.73	0	12.37	0	11.46	0	0.285	0	95	0	95	0

*Nota. Se muestra un ejemplo de cómo debe ser el formato de los archivos. Elaborado por Adnan Cáceres a través. (INAMHI, 2013).*

En la **Figura 7** se observa un ejemplo del envío de 5 parámetros hidrometeorológicos con su respectiva calidad de dato, a continuación, se enlistan cada uno de los campos del archivo con su respectiva descripción:

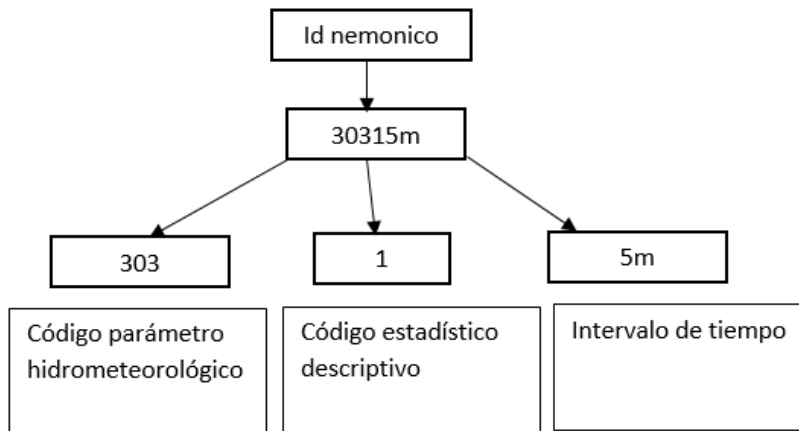
- **Header.** Corresponde a la primera línea de cada archivo e identifica que datos van a ir en cada columna del archivo (INAMHI, 2013).
- **fecha.** Como primer campo debe ir escrita la palabra fecha, esto indica que en la primera columna se va enviar el valor de la fecha de toma del dato (INAMHI, 2013).

- **Etiqueta parámetro.** Después del tercer campo se deben enviar las etiquetas que representan a cada parámetro hidrometeorológico observado, esta etiqueta está compuesta por el Id sensor y por el Id nemónico (INAMHI, 2013).

El id nemónico es un campo compuesto que está conformado de la siguiente manera en la **Figura 8**, donde, los primeros 3 dígitos corresponden al código del parámetro hidrometeorológico observado, el cuarto dígito corresponde al código estadístico descriptivo y el último dígito corresponde al intervalo de tiempo y la unidad de medida del tiempo.

### Figura 8

*Estructura del Id Nemónico.*



*Nota. ID nemónico es un campo compuesto. Elaborado por Adnan Cáceres a través. (INAMHI, 2013).*

**Payload.** Corresponde a los valores de los parámetros indicados en la primera fila (INAMHI, 2013).

**Valor fecha.** Contiene el valor de la fecha y hora en formato YYYYMMDDHHMM en la que fue tomado el dato (INAMHI, 2013).

**Valor parámetro.** Este campo representa la medida que tomó el sensor del parámetro indicado (INAMHI, 2013).

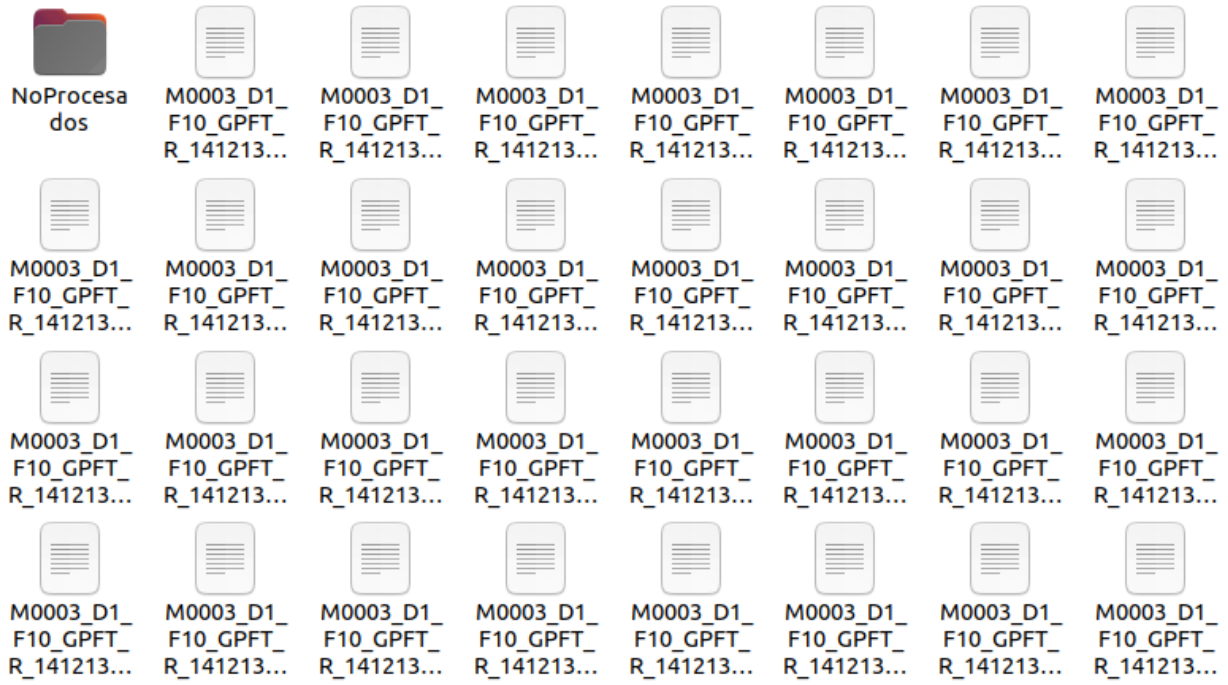
**Valor de calidad dato.** Corresponde al calificativo que asigna el sensor a la magnitud observada (INAMHI, 2013).

### ***1.6 Archivos no Procesados.***

En los servidores del INAMHI cada día se crea una nueva carpeta con los archivos (. rep), no obstante, en ocasiones existen archivos que por diversas razones no se procesan, por ello, se crea de manera automática una carpeta con el nombre **NoProcesados** como se muestra en la **Figura 9**, cabe aclarar que esta carpeta es creada por la institución, así que, no se los toma en cuenta para la elaboración del proyecto.

### **Figura 9**

*Carpeta de Archivos no procesados.*



*Nota. Carpeta de archivos no procesados junto a archivos que se pueden procesar. Elaborado por Adnan Cáceres.*

### 1.7 Umbrales para los archivos (. rep).

Los umbrales son los valores máximos y mínimos que pueden llegar a tener las variables que se encuentran en cada archivo (. rep). Así que, se realiza la matriz de la **Figura 10**. El formato del archivo se compone por el nombre de la estación seguido del nombre del *Datalogger*.

**Figura 10**

*Estructura de la Matriz umbrales.*

Cont	1	2	3	4	5
M0024	544161m	29341m	29311m	29321m	293321m
Oper	nada	promedio	suma	promedio	promedio
Op	-1	0	1	0	0
UMAX	1	25.429162	24.58746	5.416294	78.582875
UMIN	0	9.570837	9.612539	-5.204294	49.417124

*Nota. Matriz que contiene umbrales máximo y mínimo. Elaborado por Adnan Cáceres a través de (INAMHI, 2013).*

A continuación, se describe el contenido de cada fila:

- Primera fila contiene un contador para saber cuántas columnas con las que se trabaja tiene el archivo.
- Segunda fila contiene el índice o cabeceras.
- Tercera fila índice de operación por columna a realizar en el archivo a procesar 0=promedio, 1=suma, nada=-1.
- Cuarta fila tiene umbral máximo de cada columna.
- Quinta fila tiene umbral mínimo a cada columna.
- Sexta fila valor a remplazar en caso de no cumplir con el rango max-min.

### ***1.8 Metodologías Ágiles***

La utilización de metodologías ágiles facilita el manejo de proyectos de desarrollo de *software* debido a las iteraciones rápidas, es decir, las metodologías en el desarrollo ágil son de gran ayuda por diversifican el trabajo ayudando a construir *softwares* en poco tiempo y así aumentando la satisfacción de los clientes. Al desarrollar este proyecto se usó la metodología de desarrollo SCRUM, ya que, al ser una metodología ágil, está abierto a nuevos requerimientos del proyecto y permite una colaboración directa con los interesados. En este caso el departamento de información del INAMHI. SCRUM al centrarse en la mejora continua, principio fundamental de la metodología ágil, ayudara la adaptación de los factores fluctuantes que no son cubiertos al 100% en los requerimientos iniciales del desarrollo del programa.

### **1.8.1 SCRUM.**

Considerado una metodología de desarrollo incremental iterativo debido a los entornos complejos, donde los requisitos son cambiantes o no están muy bien definidos en su totalidad (Gallego, 2012).

### **1.8.2 Roles Scrum.**

- **Product Owner:** Persona que representa al cliente, es decir, es el encargado de ver todas las necesidades del usuario para posteriormente comunicar al scrum máster y al *team development*, para que conjuntamente desarrollen dicha necesidad (Gallego, 2012).
- **Scrum Máster:** Es el moderador entre el *Product Owner* y el *Team development*, encargado de eliminar dificultades que afecten a la entrega del producto además gestiona el proceso Scrum, también facilita reuniones y eventos si lo considera pertinente (Gallego, 2012).
- **Development team:** Personas altamente capacitadas para dar solución a la necesidad o requisitos del cliente, el equipo suele estar formado de entre 3-9 personas auto-organizados y auto-gestionándose para proporcionar un entregable.
- **Usuarios:** Personas que usaran el producto (Gallego, 2012).
- **Product Backlog:** Lista de requerimientos obtenidos del cliente por parte del *Product Owner*, que darán cumplimiento a la solicitud del cliente (Gallego, 2012).
- **Sprint Planning Meeting:** Es la primera reunión que se lleva a cabo para planificar el entregable de la primera fase hasta el final del producto, es decir, se obtiene una lista de funcionalidades tomadas el *Product Backlog* (Gallego, 2012).



- ***Sprint Backlog***: Es la lista de funcionalidades obtenidas del *Product Backlog*, es decir, es el conjunto de requisitos que se deben presentar en un tiempo estimado de 1-4 semanas más conocidas como *Sprint* (Gallego, 2012).
- ***Sprint***: Es el proceso de desarrollo de la necesidad del cliente, divididas en un módulo funcional incremental, aquí intervienen el *Scrum Master* y el *Development Team* (Gallego, 2012).
- ***Dayli Scrum***: Son reuniones que se realizan diariamente para dar seguimiento a todos los procesos que se tengan dentro del *Sprint* donde se reúnen el *Scrum Máster* y el *Development Team* y se realizan preguntas muy puntuales, la reunión dura de 8-15 minutos (Gallego, 2012).

## CAPÍTULO II

### 2 ANÁLISIS Y REQUERIMIENTOS.

Este capítulo describe el análisis realizado para recopilar los requisitos del sistema, incluyendo el análisis de factibilidad y los requisitos funcionales y no funcionales necesarios obtenidos de las reuniones realizadas por el INAMHI.

#### *2.5 Análisis de factibilidad.*

La factibilidad contiene tres aspectos necesarios para el análisis de un proyecto las cuales ayudaron a la toma de decisión para llevar a cabo el mismo estas son: viabilidad técnica, económica y operacional

*2.1.1 Viabilidad Técnica.* Para el desarrollo del presente proyecto se utilizarán tecnologías *open source* y actuales que permitan acceder al servicio sin ningún problema, el sistema está considerado escalable debido a que son módulos que se incorporan con otras para cumplir las diferentes necesidades que requiere la institución.

Los requisitos de hardware establecidos por este trabajo se muestran en la **Tabla 4.**

**Tabla 4**

*Requisitos del Hardware.*

Requisitos del <i>hardware</i>			
Dispositivo	Cantidad	Uso	Descripción
Monitor	1	Desarrollo, investigación, pruebas y documentación	Monitor Samsung de 32”.
CPU	2	Desarrollo, investigación, pruebas y documentación	-Procesador: AMD Ryzen5.  - Procesador i5 7ma generación  - Memoria RAM: 32 GB  - Memoria RAM: 8GB  -Discos: 2TB.  - Arquitectura: 64 Bits.
Router	1	Red de conexión a internet.	-Velocidad: 40 MB

*Nota. Requisitos del hardware para el desarrollo del proyecto. Elaborado por: Adnan Cáceres.*

Por otro lado, los requisitos de *software* establecidos en este trabajo se muestran en la **Tabla 5**.

**Tabla 5**

*Requisitos del Software.*

Requisitos del <i>software</i>			
Producto	Cantidad	Uso	Descripción
Windows 10	1	Sistema operativo	-licencia: Educativo. -Año: 2021.
PostgreSQL	1	Software de Base de Datos	-Licencia: GPL -Versión: 14
Python	1	Lenguaje de programación	-Licencia: GPL -Versión: 3
Virtual Box	1	Software de virtualización	-Licencia: GPL -Versión: 6.1
Visual Studio Code	1	Editor de código	-Licencia: GPL -Versión: 1.6
Git Hub	1	Repositorio de código y versionamiento.	-Licencia: GPL.
Google Chrome	1	Explorador web	-Licencia: GPL
Google Meet	1	Herramienta para reuniones virtuales.	-Licencia: Gratuita
Zoom	1	Herramienta para reuniones virtuales.	-Licencia: Gratuita
Microsoft Office	1	Herramienta para documentación	-Licencia: Educativo

*Nota. Requisitos del software para el desarrollo del proyecto. Elaborado por: Adnan Cáceres.*

**2.1.2 Viabilidad Económica.** Al haber tenido un análisis previo por el INAMHI, el proyecto es rentable. El costo de los equipos de *hardware* está cubierto debido a que los equipos se adquirieron con anterioridad. La mayoría del *software* usado en el proyecto es *open source* a excepción del SO Windows 10 que está bajo una licencia educativa. Otros gastos operativos como transporte e internet se cubren por el autor del proyecto como se puede ver en la **Tabla 6**.

**Tabla 6**

*Costos del desarrollo del Programa.*

Recurso	Descripción	Cantidad	Precio unitario	Total
Sistema Operativo	Windows 10	1	\$8.45	\$8.45
Office	Microsoft Office	1	\$33.5	\$33.5
Lenguajes de Programación	<i>Open Source</i>	1	\$0	\$0
Github	<i>Open Source</i>	1	\$0	\$0
PostgreSQL	<i>Open Source</i>	1	\$0	\$0
<i>Visual Studio Code</i>	<i>Open Source</i>	1	\$0	\$0
<i>Virtual Box</i>	<i>Open Source</i>	1	\$0	\$0
Total				\$41.95

*Nota. Costos totales del desarrollo del Programa. Elaborado por: Adnan Cáceres*

Los costos del *hardware* se muestran en la **Tabla 7**.

**Tabla 7***Costos de Hardware.*

Recurso	Descripción	Cantidad	Precio unitario	Total
PC escritorio	Componentes descritos en la Tabla 4	1	\$1500	\$1500
Portátil	Componentes descritos en la Tabla 4	1	\$500	\$500
Monitor	Monitor Samsung	1	\$250	\$250
Total				\$2250

*Nota. Costos totales de Hardware. Elaborado por: Adnan Cáceres*De igual forma, los costos de los servicios se detallan en la **Tabla 8**.**Tabla 8***Costos de Servicios.*

Recurso	Descripción	Cantidad	Precio unitario	Total
Servicio de Internet	Internet de 40MB	1 por mes duración total 6 meses	\$24	\$144
Servicio de transporte	autobuses, Uber	2 por reunión total reuniones 10 2 por reunión	\$0.25  \$7	\$5  \$14
Total				\$163

*Nota. Costos totales por los diferentes servicios. Elaborado por: Adnan Cáceres*

La **Tabla 9** muestra los costos totales para la ejecución del proyecto.

**Tabla 9**

*Costo de recursos.*

Referencia	Valor
Tabla 3: Costos de <i>Software</i>	\$41.95
Tabla 4: Costos de <i>Hardware</i>	\$ 2250
Tabla 5: Costos de Servicios	\$163
Total	\$2454.95

*Nota. Costo del proyecto tomando en cuenta los costos de hardware, software y servicios. Elaborado por: Adnan Cáceres*

**2.1.3 Viabilidad Operacional.** En la actualidad, el INAMHI cuenta con un proceso de datos que provienen de las estaciones automáticas, este proceso no cumple con los requerimientos que la institución requiere. Por esta razón, se propone el desarrollo de este proyecto para dar solución al problema. Además, los usuarios que manejan esta información son profesionales y expertos en el procesamiento de datos que al recibir los archivos depurados de las estaciones automáticas ya procesados podrán utilizarlos de forma más eficaz.

## **2.6 *Análisis de Requerimientos.***

En este subcapítulo se describe de una manera más detallada tanto los requerimientos técnicos como de *software* del programa y la funcionalidad del mismo.

### **2.2.1 *Alcance.***

Este trabajo implementará técnicas basadas en procesamiento distribuido, con una previa investigación para seleccionar una metodología adecuada que permita plantear una alternativa de solución al problema, tomando en consideración las limitantes tanto de *software* como de *hardware* y buscando un balance entre ambas. El programa desarrollado es de uso exclusivo para el INAMHI, ya que contempla el procesamiento de los archivos fuente receptados en los servidores de esta institución. El proceso termina cuando los datos son almacenados adecuadamente en la base de datos, luego se procede a una fase de validación y corrección de errores no contemplada en este proyecto.

### **2.2.2 *Requerimientos técnicos.***

Los requisitos técnicos se refieren al diseño e implementación del *software*, estos se pueden ver a detalle en la **Tabla 10**.



**Tabla 10**

*Requerimientos técnicos.*

Tipo de Requerimiento	Descripción
Idioma de Programación	Phyton
Sistema Operativo	Linux
Normas	Normas establecidas por el INAMHI F10

*Nota. Se muestra el lenguaje de programación en el que el proyecto se desarrolló, el sistema operativo y las normas que son establecidas por la institución. Elaborado por: Adnan Cáceres*

## CAPÍTULO III

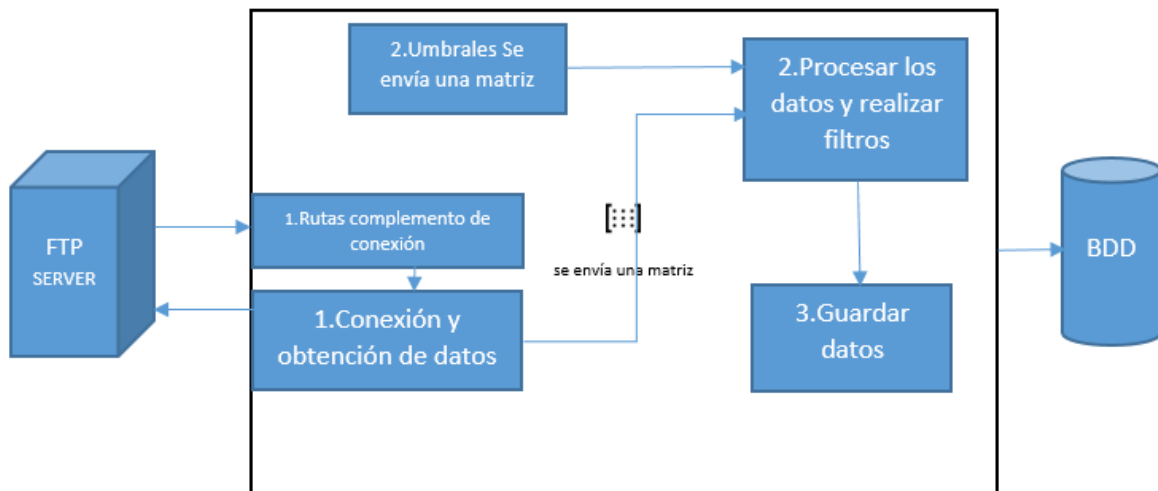
### 3 DESARROLLO

#### 3.5 Arquitectura

Arquitectura de módulos, en la **Figura 11** se observa los módulos que el proyecto va a presentar.

**Figura 11**

*Arquitectura de los módulos del proyecto.*



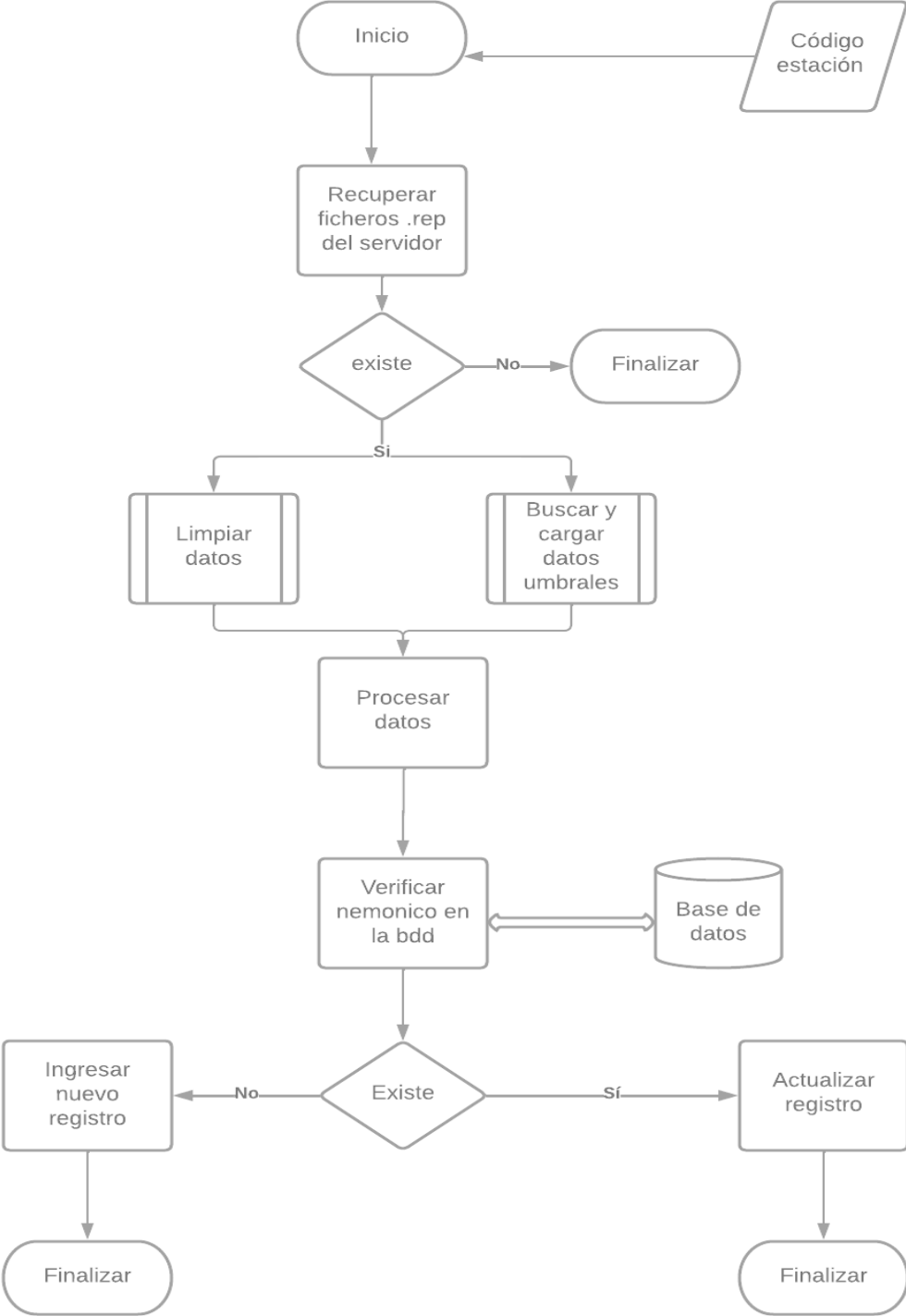
*Nota. Se muestra los módulos que se van a trabajar en el proyecto. Elaborado por Adnan Cáceres.*

En la **Figura 11** lo primero que se realiza es ir al servidor FTP en la cual se obtiene unos links, en base a esos links se busca en el servidor los ficheros que tiene los archivos .rep, si el fichero no existe no trae nada, es decir, termina el subproceso, pero el resto de subprocesos siguen funcionando, una vez se obtienen los datos se envía una matriz tanto del módulo de Conexión como del módulo de Umbrales hacia el módulo donde se procesa y filtra los datos, de esta forma, nos dirigimos al módulo guarda datos, para finalmente guardar los datos en la BDD.

A continuación, en la se muestra el diagrama de flujo mediante el cual se da una breve idea de cómo fue desarrollado el proyecto.

Figura 12

Diagrama de flujo.



*Nota. Aquí se observa como la arquitectura de flujo de datos se implementa en el proyecto. Elaborado por Adnan Cáceres.*

Para la elaboración del proyecto se necesita tener un servidor FTP, por ello, mediante el uso de *Virtual Box* se crea una máquina virtual con el Sistema Operativo Linux (Ubuntu 20.04), en el cual se instaló Ubuntu Server para realizar la simulación del servidor FTP que tienen en el INAMHI. En el **Anexo 1** se muestra el estado del servidor FTP.

Una vez creado el servidor, se procede a guardar en una carpeta los archivos (.rep) proporcionados por la institución (INAMHI), los cuales, mediante el comando *tree* seguido del nombre de la carpeta que contiene los archivos. La organización interna de los ficheros se muestra en el **Anexo 2**.

### ***3.6 Desarrollo del programa en Phyton3.***

En primer lugar, se realiza la creación del paquete **obtener\_datos** en el cual se crean las clases que se muestran en el **Anexo 3**, con esas clases podemos recuperar los ficheros .rep del servidor FTP (al no tener acceso al servidor físico del INAMHI se implementa rutas por defecto), en caso de que no exista el archivo el subprocesso termina, caso contrario se realizan dos subprocessos de forma simultanea los mismos que se encuentran en el paquete **paralelismo**, clase **ProcesoParalelo.py**, el cual se observa en el **Anexo 4** Paquete paralelismo.

#### ***3.6.1 Limpieza de los datos obtenidos del servidor FTP***

Una de las principales secciones del proyecto es la limpieza de los datos obtenidos del servido ftp, teniendo en cuenta varios factores a evaluar y controlar, tales como eliminar caracteres extraños y verificar el número de filas que contiene cada

fichero, de esta manera, garantizando que los datos utilizados en la sección de procesamiento de datos no presenten errores por realizar operaciones matemáticas.

### Figura 13

*Limpieza de los datos obtenidos del servidor FTP.*

#### ALGORITMO (limpiar)

Variables:

Entero: tamaño\_umbral

Array: datos, datos\_limpios

Inicio

    SI (tamaño\_umbral== tamaño\_datos)

        datos\_limpios = eliminar\_caracteres\_especiales(datos)

    De lo contrario

        Salir

Fin\_Inicio

Fin (limpiar)

*Nota. Pseudocódigo de la limpieza de los datos obtenidos del servidor FTP. Elaborado por: Adnan Cáceres.*

#### **3.6.2 Buscar y cargar los datos umbrales.**

La sección está encargada de buscar los archivos respectivos que contienen los umbrales de cada estación, para preceder a cargarlos en una matriz y enviarlos al módulo de procesamiento, para esto, es necesario obtener el nombre de la estación con la cual se está trabajando, si el archivo de umbral no es encontrado se procede a finalizar ese proceso

## Figura 14

*Buscar y cargar datos umbrales.*

### ALGORITMO (cargar)

Variables:

String: nombre\_archivo

Array: datos

Inicio

SI (existe(nombre\_archivo))

datos = cargar\_datos(datos)

De lo contrario

Salir

Fin\_Inicio

Fin (cargar)

*Nota. Pseudocódigo de la búsqueda y carga de los datos umbrales. Elaborado por: Adnan Cáceres.*

Cada subproceso envía una matriz para poder procesar los datos, una vez procesados los datos verificamos mediante el Id Nemónico ( **Figura 8**) en la base de datos si existe o no.

En el caso de que Exista se actualiza el registro y se finaliza el proceso.

Si no existe se ingresa un nuevo registro y se finaliza el proceso.

## CAPÍTULO IV

### 4 PRUEBAS Y RESULTADOS

Para la obtención de resultados de rendimiento de los módulos del proyecto se utilizó una PC con las siguientes características: 8 GB de memoria RAM, un procesador i5 de séptima generación con 4 núcleos a 2.5 GHz.

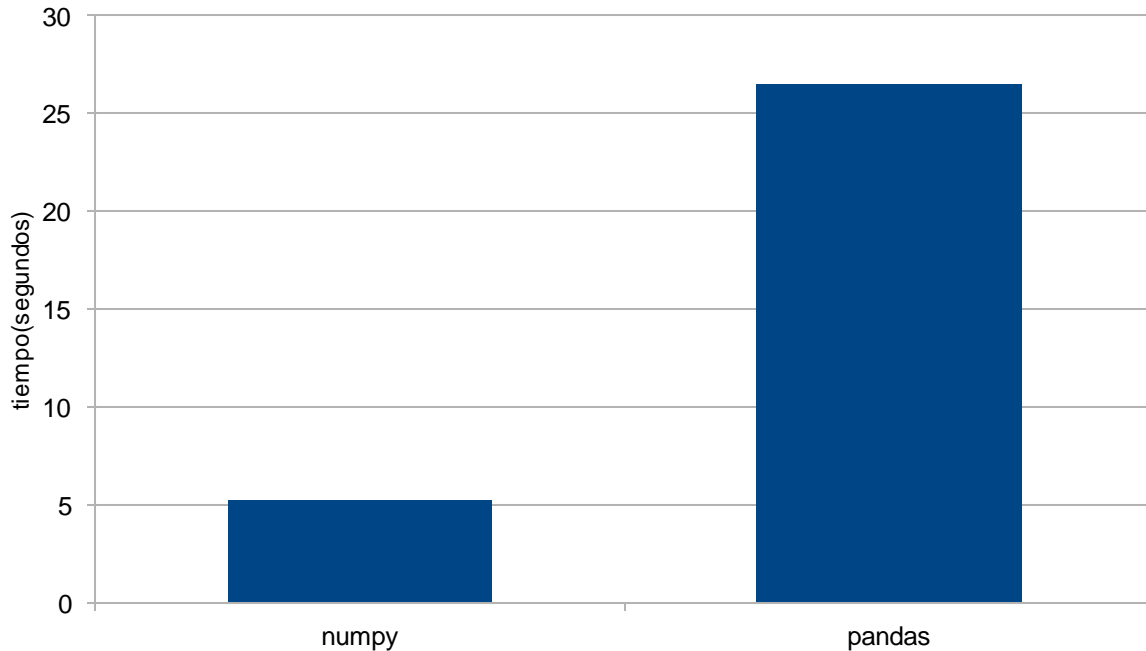
#### *4.1 Comparación de rendimiento de librerías para procesamientos de datos en Python.*

La librería con mejor tiempo de respuesta es Numpy con 5.19 s frente a Pandas que demora 26.5 s en procesar 5000 archivos en un monoproceso, presentando un rendimiento 79% más eficiente.

Como se puede observar en la **Figura 15**.

#### **Figura 15**

*Tiempo promedio Numpy vs Pandas.*



Nota. Comparación de rendimiento de librerías Pandas vs Numpy. Elaborado por Adnan Cáceres.

#### ***4.2 Módulos y ficheros de prueba.***

Para realizar métricas sobre pruebas de rendimiento para obtener tiempo de ejecución se utilizó módulos, modulo 1 encargado de conectarse al servidor ftp buscar el archivo de una estación y recuperar el archivo, modulo 2 consiste en limpiar y filtrar los datos obtenidos del módulo y realizar el procesamiento de datos y el módulo 3 encargado de gestionar los resultados del módulo 2 y guardarlos en una base de datos.

Las pruebas de rendimiento se realizaron con 3000 archivos en dos escenarios, en monoproceso ejecutar el proyecto de forma secuencial y el multiproceso que ejecuta todo el proceso definido por los módulos en subprocesos, para la prueba se ejecutó sobre 15 subprocesos.

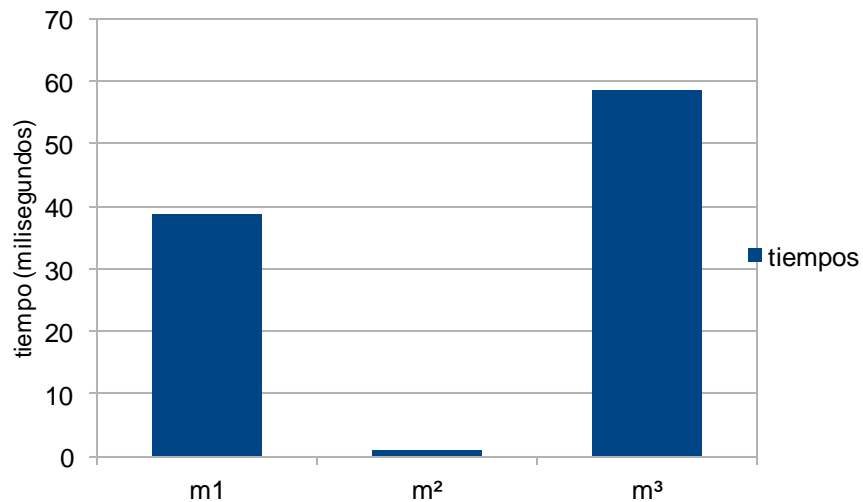


#### 4.2.1 Métricas de rendimiento monoproceso.

Los módulos con mayor tiempo de ejecución son el módulo 3 con 38.9 milisegundos y el módulo 1 con 38.9 milisegundos, siendo el módulo 2 quien presenta mejor rendimiento durante toda la ejecución del proyecto. Como se observa en la **Figura 16**.

**Figura 16**

*Tiempo de ejecución monoproceso.*

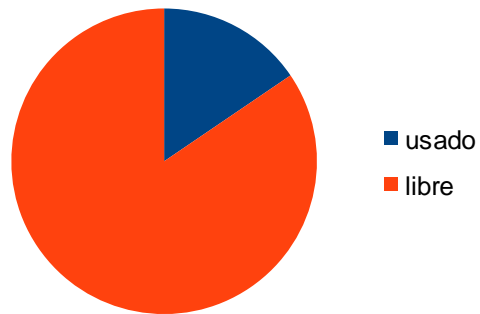


Nota. Tiempo promedio en milisegundos de los tres módulos. Elaborado por Adnan Cáceres.

La evaluación de rendimiento de CPU sobre la aplicación en monoproceso muestra un consumo medio de 15.5% y un máximo de 27,6% durante la ejecución del proyecto tal como se observa en la **Figura 17**.

**Figura 17**

*Consumo de CPU en monoproceso.*

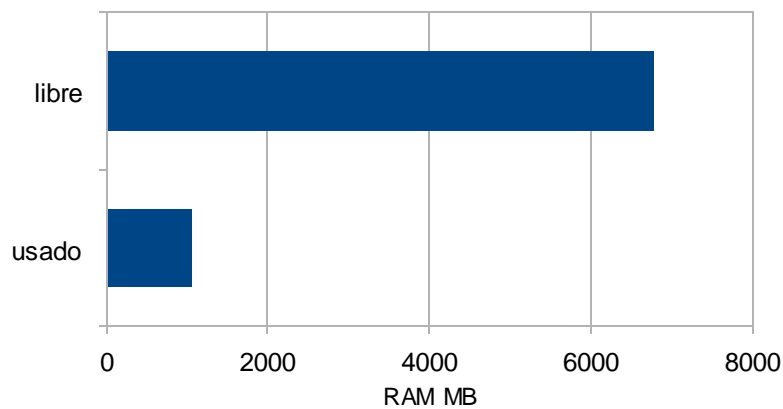


Nota. Consumo promedio de CPU. Elaborado por Adnan Cáceres.

El consumo de memoria RAM tiene un promedio de 1051 Mb llegando a un máximo de 1072.6 ocupando un 13.7% de total de memoria, tal y como se muestra en la **Figura 18**.

**Figura 18**

*Consumo de RAM en monoproceso.*



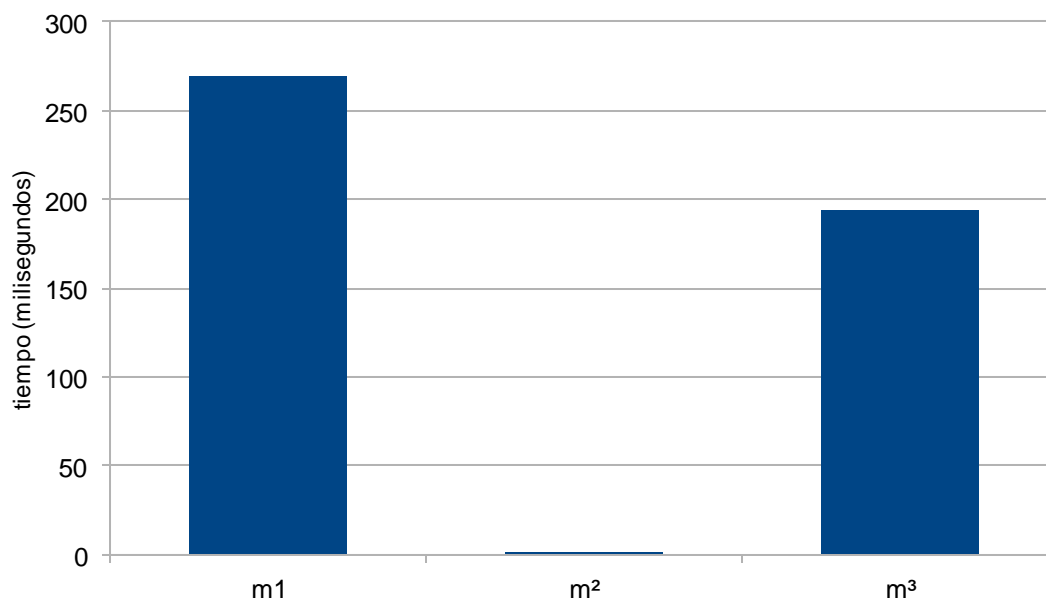
Nota. Consumo de memoria RAM en MB. Elaborado por Adnan Cáceres.

#### 4.2.2 Métricas de rendimiento multiprocesos.

Los módulos con mayor tiempo de ejecución son el módulo 1 con 270 milisegundos y el módulo 3 con 194 milisegundos, siendo el módulo 2 quien presenta mejor rendimiento durante toda la ejecución del proyecto. Como se observa en la **Figura 19**.

**Figura 19**

*Tiempo de ejecución multiproceso.*

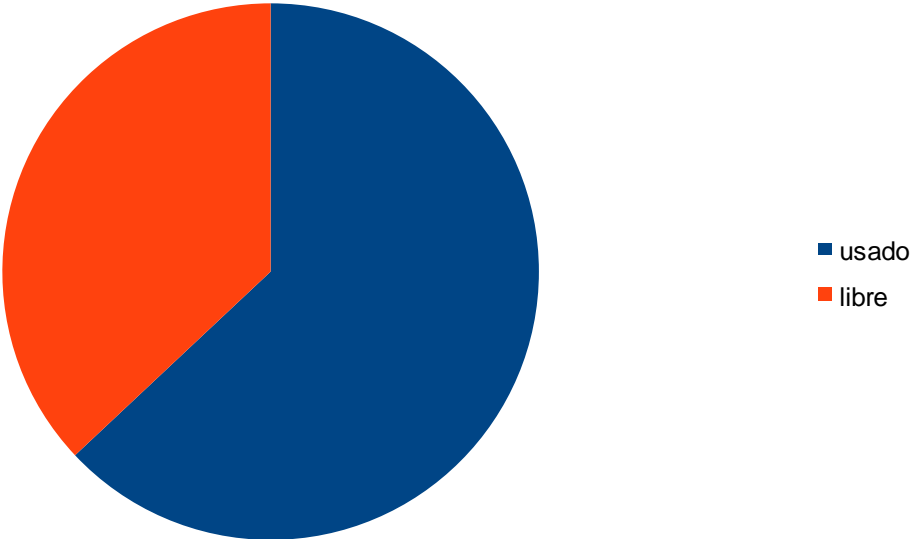


Nota. *Tiempo promedio de los módulos en multiproceso. Elaborado por Adnan Cáceres.*

El consumo de CPU es superior al monoproceso dando como promedio el 63% de uso, llegando a un máximo de 68.9%, tras ejecutar todo el proyecto, tal como se muestra en la **Figura 20**.

**Figura 20**

*Consumo de CPU en multiproceso.*

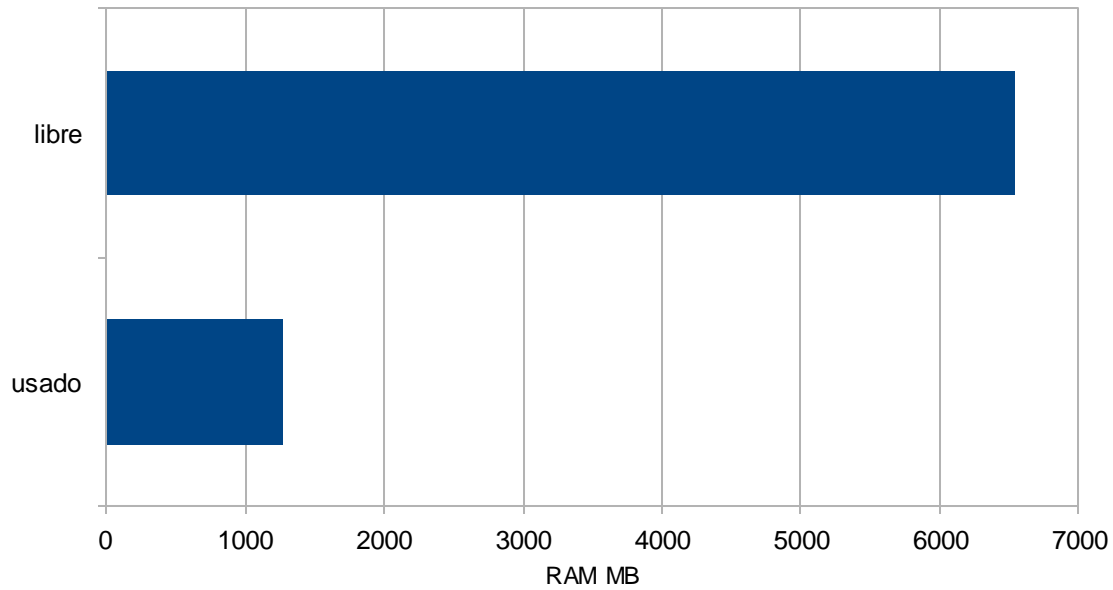


*Nota. Uso de CPU en tareas multiproceso. Elaborado por Adnan Cáceres*

El consumo de memoria RAM tiene un promedio de 1274.6 Mb llegando a un máximo de 1330.4 ocupando un 16.2% de total de memoria, tal y como se muestra en la **Figura 21**.

**Figura 21**

*Consumo de RAM en multiproceso.*



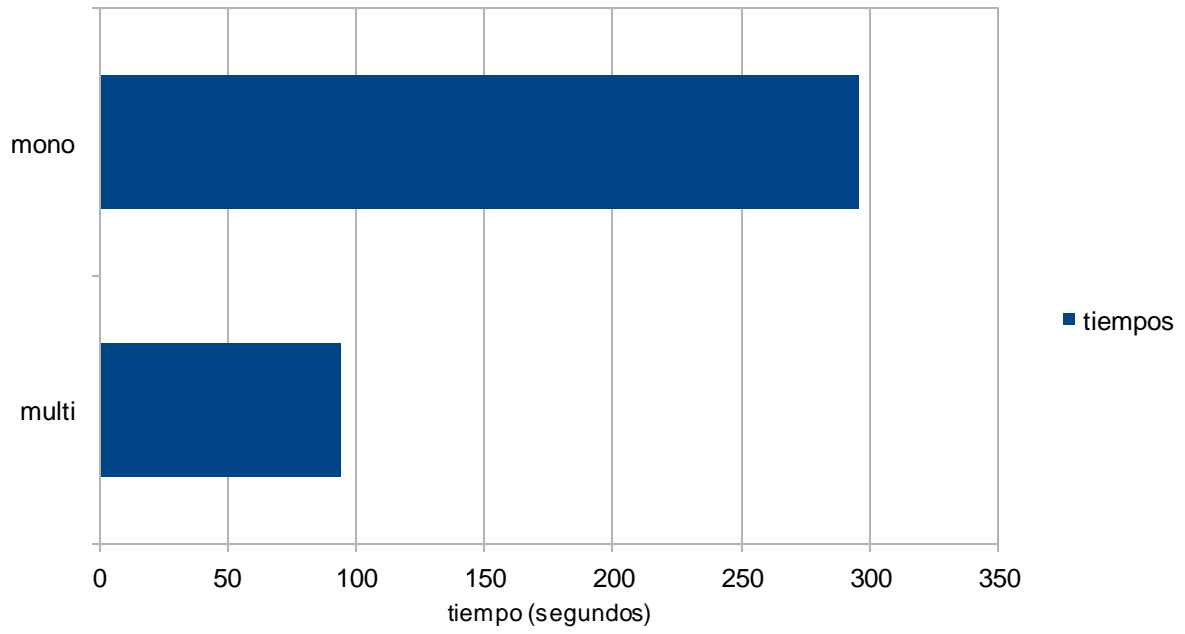
Nota. Consumo de RAM en multiprocesos. Elaborado por Adnan Cáceres.

#### 4.2.3 Comparación Tiempo de ejecución entre monoproceso y multiproceso

El tiempo final tras ejecutar 3000 archivos en monoproceso se demora 295.8 s frente a los 94.5 s en multiprocesos, siendo este último superior con un 100% frente al monoproceso. Como se observa en la **Figura 22**.

#### Figura 22

*Tiempo de ejecución multiproceso vs monoproceso.*



*Nota. Tiempo total de ejecución. Elaborado por Adnan Cáceres.*

## CONCLUSIONES

En este proyecto se desarrolló un programa para el procesamiento de archivos fuente de datos que provienen de las estaciones hidrometeorológicas automáticas del INHAMI. Gracias al análisis del formato estándar F10 se logró la comprensión de la estructura de los datos y sus *datalogger*.

El uso de los *scripts*, para verificar el estado de los archivos del servidor, permitió reducir la frecuencia de llegada de los ficheros. Además, se logró filtrar los archivos fuente con errores, mitigando la congestión del servidor y evitando el uso de recursos computacionales innecesarios.

Para el desarrollo de los *scripts* se utilizaron herramientas de software libre, dando cumplimiento a uno de los requerimientos del INAMHI. En este sentido, las librerías de Python como Numpy y Pandas, fueron de mucha utilidad. Siendo Numpy mucho más eficiente en cuanto al tiempo de procesamiento.

El tiempo final tras ejecutar 3000 archivos en monoproceso se demora 295.8 s frente a los 94.5 s en multiprocesos, siendo este último superior con un 110% frente al monoproceso, por ende, el uso de multiprocesos es fiable y muy superior frente al monoproceso.

Considerando que el volumen de archivos receptados en el servidor cada cinco minutos, está en el orden de miles, fácilmente el volumen de información se puede convertir en un problema de BIG DATA. Por lo tanto, se implementó programación paralela para disminuir el tiempo de procesamiento de los archivos y aprovechar de mejor manera los recursos del servidor.

## RECOMENDACIONES

Se recomienda que el INAMHI mejore la gestión de los archivos (.rep), ya que algunos de ellos presentaron errores como falta de datos o datos en otros formatos, lo que complica la carga de la información. Si se realiza una buena selección de los ficheros, se podrían cargar los datos directamente como una matriz, facilitando las operaciones.

También es recomendable, tener una buena gestión de los archivos históricos, ya que en su mayoría no se encuentran depurados.

Para el uso de multiprocesos, el hardware que se usa debe tener una memoria RAM aceptable al igual que su procesador, en este caso se usó una memoria de 8 GB de memoria RAM, procesador i5 7ma generación de 4 núcleos a 2.5 GHz.



## REFERENCIAS

- Bernal, F., Albarracín, C., Gaona, J., & Nieto, J. (s.f.). *ferestrepoca*. Obtenido de [http://ferestrepoca.github.io/paradigmas-de-programacion/paralela/paralela\\_teoría/index.html#twelve](http://ferestrepoca.github.io/paradigmas-de-programacion/paralela/paralela_teoría/index.html#twelve)
- Date, C. (2001). *Introducción a los sistemas de bases de datos*. Pearson Educación.
- devCamp. (2020). *devCamp by Bottega*. Obtenido de <https://devcamp.es/que-es-libreria-programacion/#:~:text=En%20este%20sentido%2C%20una%20librer%C3%ADa,llanamente%2C%20es%20un%20archivo%20importable.>
- eIFaroStudio. (s.f.). *todoxampp*. Obtenido de <https://todoxampp.com/mongodb-o-postgresql-cual-debo-utilizar/>
- EUROINNOVA. (s.f.). *EUROINNOVA INTERNATIONAL ONLINE EDUCATION*. Obtenido de <https://www.euroinnova.edu.es/blog/herramientas-de-programacion>
- Federico Di Gregorio, D. V. (29 de 12 de 2021). *The Psycopg Team*. Obtenido de <https://www.psycopg.org/docs/>
- Final Test. (s.f.). *Final Test*. Obtenido de [https://www.finaltest.com.mx/product-p/art-4.htm#:~:text=Un%20registrador%20de%20datos%20\(data%20logger,todos%20est%C3%A1n%20basados%20en%20microcontroladores.](https://www.finaltest.com.mx/product-p/art-4.htm#:~:text=Un%20registrador%20de%20datos%20(data%20logger,todos%20est%C3%A1n%20basados%20en%20microcontroladores.)
- Gallego, M. T. (2012). *Metodología scrum*.
- Gustavo B. (8 de marzo de 2021). *Hostinger*. Obtenido de <https://www.hostinger.es/tutoriales/que-es-github>
- Hidrología, I. N. (2014). *Anuario Meteorológico*. Quito: Dirección Ejecutiva del INAMHI.
- Hidrología, I. N. (2021). *Anuario Meteorológico*. Quito: Dirección Ejecutiva del INAMHI.
- historiadelaeempresa. (s.f.). *historiadelaeempresa.com*. Obtenido de <https://historiadelaeempresa.com/multithreading-vs-multiprocessing>
- INAMHI, P. (2013). *Propuesta para el formato de archivo de las estaciones automáticas que transmiten por GPRS*. Quito.
- Ivet Challenger, Diaz, Y., & Becerra, R. (2014). El lenguaje de programación Python/The programming language Python. *Ciencias Holguín*, 14.
- J. Postel, J. R. (1985). *File Transfer Protocol*. California.
- JetBrains. (2022). *JetBrains*. Obtenido de <https://www.jetbrains.com/es-es/pycharm/>

- M, P., L, R., & F, F. (2017). *Administración de base de datos con PostgreSQL*. Alicante: Editorial Área de Innovación y Desarrollo, S.L.
- Marco A. Peña Basurto, J. M. (2001). *Introducción a la programación en C*. Barcelona : edicionsupc.
- Microsoft. (s.f.). *visual studio code*. Obtenido de <https://code.visualstudio.com/docs>
- Montoro, F. (2012). *Python 3 al descubierto* . Madrid: rclibros.
- Mundial, O. M. (2010). *Guía de Instrumentos y Métodos de Observación Meteorológicos*. Ginebra.
- Mundial, O. M. (2010). *Guía del Sistema Mundial de Observación*. Ginebra.
- Mundial, O. M. (2010). *Guía del Sistema Mundial de Observación*. Suiza.
- Oracle. (s.f.). *OCI*. Obtenido de <https://www.oracle.com/big-data/what-is-big-data/>
- OSWALDO, B. C. (Octubre de 2021). *CLASIFICACIÓN AUTOMÁTICA DE TEXTO EN VARIABLES PEST A TRAVÉS DE REPRESENTACIONES VECTORIALES*. Guayaquil.
- Postel, J. (1981). *Transmission Control Protocol*. California.
- Proviento. (2014). *Proviento S.A.S*. Obtenido de <http://www.proviento.com.co/proyectos.html>
- Python Software Foundation. (s.f.). *Python* . Obtenido de <https://docs.python.org/3/library/ftplib.html>
- Roberto Solé. (11 de 11 de 2020). *hardwaresfera*. Obtenido de <https://hardwaresfera.com/articulos/que-es-virtualbox/>
- Sarasa, A. (2016). *Introducción a las bases de datos NoSQL usando MongoDB*. Barcelona: UOC.
- Sierra, F. J. (2015). *Java 2: lenguaje y aplicaciones* . Madrid: RA-MA Editorial.
- SMART LIFE. (s.f.). *SMART LIFE*. Obtenido de <https://www.surosystem.com/partners-oracle>
- the pandas development team. (2008). *pandas*. Obtenido de [https://pandas.pydata.org/docs/getting\\_started/overview.html](https://pandas.pydata.org/docs/getting_started/overview.html)
- tutorialspoint. (s.f.). *tutorialspoint*. Obtenido de [www.tutorialspoint.com/python3/time\\_time.htm#](http://www.tutorialspoint.com/python3/time_time.htm#)
- Valenzuela, J. S. (2018). *Python: aplicaciones practicas*. Madrid: RA-MA Editorial.

## Anexo 1 Estado del servidor FTP

```
adnan@adnan-VirtualBox:~$ systemctl status vsftpd.service
● vsftpd.service - vsftpd FTP server
   Loaded: loaded (/lib/systemd/system/vsftpd.service; enabled; vendor preset>
   Active: active (running) since Mon 2022-06-06 21:28:07 -05; 3min 18s ago
   Process: 764 ExecStartPre=/bin/mkdir -p /var/run/vsftpd/empty (code=exited,>
  Main PID: 765 (vsftpd)
     Tasks: 1 (limit: 16725)
    Memory: 708.0K
   CGroup: /system.slice/vsftpd.service
           └─765 /usr/sbin/vsftpd /etc/vsftpd.conf

jun 06 21:28:07 adnan-VirtualBox systemd[1]: Starting vsftpd FTP server...
jun 06 21:28:07 adnan-VirtualBox systemd[1]: Started vsftpd FTP server.
```

Anexo 2 Organización interna de los ficheros.

```
adnan@adnan-VirtualBox:~$ tree regiones/
regiones/
├── Costa
│   ├── M0162
│   │   ├── M0162_D1_F10_GPFT_R_140721213005.rep
│   │   └── M0162_D1_F10_GPFT_R_220223203504.rep
│   ├── M0168
│   │   ├── M0168_D1_F10_GPFT_R_140809124004.rep
│   │   └── M0168_D1_F10_GPFT_R_220223050005.rep
│   └── M1233
│       ├── M1233_D1_F10_GPFT_R_140721213005.rep
│       └── M1233_D1_F10_GPFT_R_220223050004.rep
├── Oriente
│   ├── M0008
│   │   ├── M0008_D1_F10_GPFT_R_151204155504.rep
│   │   └── M0008_D1_F10_GPFT_R_210213050005.rep
│   ├── M0563
│   │   ├── M05631045_2015_09_14_19_05
│   │   └── M05631045_2017_06_16_11_57
│   └── M5011
│       ├── M50111045_2015_09_14_18_31
│       └── M50112007_2016_05_21_21_53
└── Sierra
    ├── M0024
    │   ├── M0024_D1_F10_ETFT_R_220223050004.rep
    │   └── M0024_D1_F10_GPFT_R_150317192504.rep
    ├── M1107
    │   └── M1107_D1_F10_GPFT_R_210222050004.rep
    └── M5090
        ├── M5090_D1_F10_GPFT_R_141013202502.rep
        └── M5090_D1_F10_GPFT_R_210807185503.tmp

12 directories, 17 files
```

Anexo 3 clase ruta y clase conexión

```

class Rutas:
    nombreArchivo = "estaciones/nombres.txt"
    def obtenerFecha(self):
        return (datetime.today().strftime('%Y/%m/%d'))

    def obtenerUrls(self):
        listaRutas = []
        fecha = self.obtenerFecha()

        file = open(self.nombreArchivo, 'r')
        for linea in file:
            #listaRutas.append(linea.strip()+fecha)
            listaRutas.append(linea.strip() + "2020/10/15/")
        file.close()
        return (listaRutas)
    def rutasQuemada(self):
        lista = []
        lista.append('/home/adnan/regiones/Sierra/M1107/')
        lista.append('/home/adnan/regiones/Sierra/M0024/')
        lista.append('/home/adnan/regiones/Sierra/M5090/')

        lista2 = []
        lista2.extend(lista)
        lista2.extend(lista)
        lista2.extend(lista)
        lista2.extend(lista)
        lista2.extend(lista)
        lista2.extend(lista)
        lista2.extend(lista)
        lista2.extend(lista)
        lista2.extend(lista)
        lista2.extend(lista)
        lista2.extend(lista)
        print(str(len(lista2))+ "-----total ficheros")
        return lista

```

```

class ConexionFtp:
    ftp = None
    fullPath = []
    data = []
    info = []
    rutaDestino = "data"

    def __init__(self, host, usuario, contrasenia):
        self.host = host
        self.usuario = usuario
        self.contrasenia = contrasenia
        print("inicializacion ftp")

    def conIniciar(self):
        #print(os.path.dirname(os.path.realpath(__file__)))#ruta real

        self.ftp = FTP(self.host) # connect to host, default port
        self.ftp.login(self.usuario,self.contrasenia) # user anonymous, passwd anonymous@
        self.ftp.encoding = "utf-8"

    def conFinalizar(self):
        self.ftp.quit()

    def crearDirectorio(self):
        nombreCarpeta = self.rutaDestino
        if not os.path.exists(nombreCarpeta):
            os.makedirs(nombreCarpeta)

```

#### Anexo 4 Paquete paralelismo

