IoT-Driven Data Storage with RAID 10 and Real-Time Monitoring

A Project Report Submitted

In the partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

In

ELECTRONICS & COMMUNICATION ENGINEERING

Submitted by

AKSHAT SAXENA (2021041118)

TUSHAR GUPTA (2021041080)

ANKIT SINGH (2021041015)

SHIVANGI SHARMA (2021041074)

Under Supervision of

DR. RAJAN MISHRA (Assistant Professor)

Department of Electronics & Communication Engineering

Madan Mohan Malaviya University of Technology

Gorakhpur, Uttar Pradesh – 273010

CANDIDATE'S DECLARATION

I declare that this written submission represents my work and ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

AKSHAT SAXENA
Roll No. 2021041118
B.Tech. (ECE)
Department Of Electronics and Communication Engineering

TUSHAR GUPTA
Roll No. 2021041080
B.Tech. (ECE)
Department Of Electronics and Communication Engineering

ANKIT SINGH
Roll No. 2021041015
B.Tech. (ECE)
Department Of Electronics and Communication Engineering

SHIVANGI SHARMA
Roll No. 2021041074
B.Tech. (ECE)
Department Of Electronics and Communication Engineering

CERTIFICATE

Certified that Akshat Saxena, Tushar Gupta, Ankit Singh and Shivangi Sharma have carried out the project work presented in this report entitled "**IoT-Driven Data Storage with RAID 10 and Real-Time Monitoring**" for the award of Bachelor of Technology in Electronics and Communication Engineering from Madan Mohan Malaviya University of Technology (formerly Madan Mohan Malaviya Engineering College), Gorakhpur (UP) under my supervision and guidance. The report embodies result of original work and study carried out by students themselves and the contents of the report do not form the basis for the award of any other degree to the candidate or to anybody.

Prof. Sanjay Kumar Soni

Head of Department

Department of ECE

M.M.M.U.T. Gorakhpur

APPROVAL SHEET

This project report entitled "**IoT-Driven Data Storage with RAID 10 and Real-Time Monitoring**" by Akshat Saxena, Tushar Gupta, Ankit Singh and Shivangi Sharma is approved for the degree of Bachelor of Technology in Electronics and Communication Engineering.

Examiner

- -

Supervisor
Assistant Prof. Dr. Rajan Mishra

- -

Head of Department
Prof. Sanjay Kumar Soni

- -

ACKNOWEDGEMENT

We would like to express our profound gratitude to **Dr. Rajan Mishra (Assistant Professor, Department of Electronics and Communication Engineering)**, of Madan Mohan Malaviya University of Technology, Gorakhpur for their contributions in the completion of our project titled "**IoT-Driven Data Storage with RAID 10 and Real-Time Monitoring**".

We would like to express our special thanks to our supervisor **Assistant Prof. Rajan Mishra (Department of Electronics and Communication Engineering)** for the time and efforts he provided us throughout the year. His useful advice and suggestions were really helpful to us during the project's completion. In this aspect, we are eternally grateful to him.

Akshat Saxena (2021041118)
Tushar Gupta (2021041080)
Ankit Singh (2021041015)
Shivangi Sharma (2021041074)

TABLE OF CONTENTS

ABSTRACT

This project focuses on the development of an IoT-based storage system that leverages RAID 10 technology to offer reliable and high-performance local data storage. The system integrates ESP32-based sensors for real-time data acquisition, transmitting telemetry data to a Raspberry Pi server configured with RAID 10 for secure, redundant, and high-speed storage. The use of RAID 10 ensures data redundancy, fault tolerance, and optimized performance, making it ideal for storing large volumes of sensor data while safeguarding against potential hardware failures. Through IoT integration, the system enables seamless wireless transmission of data from the ESP32 sensors to the central server, providing real-time updates. A web-based dashboard offers a user-friendly interface for remotely monitoring and managing sensor data, allowing users to visualize data trends, track storage health, and optimize system performance. This setup eliminates reliance on cloud storage, ensuring full control over data security, access, and costs, while providing a cost-effective, scalable solution for monitoring and analyzing sensor data over time. The combination of RAID 10 and IoT technologies in this system presents an innovative approach to efficient and reliable data storage with advanced monitoring capabilities.

INTRODUCTION

In the digital age, efficient and secure data storage is crucial. While cloud storage offers scalability and accessibility, it poses challenges such as security risks, dependency on internet connectivity, and recurring costs, especially for sensitive data. Local storage solutions, particularly Network Attached Storage (NAS) with RAID (Redundant Array of Independent Disks) technology, offer enhanced security, performance, and control, ensuring data redundancy and protection. RAID 10 provides a balance of high performance and security, making it ideal for critical applications.

The Internet of Things (IoT) enables remote monitoring and management of data, offering an effective alternative to cloud storage by combining local storage with IoT technology for faster, more secure data management.

Project Scope:

This project aims to design and implement an IoT-based storage system using an ESP32 for sensor data collection, transmitted to a Raspberry Pi server with RAID 10 for secure storage. The system will include a web-based dashboard for easy management and monitoring. Key components include:

- **ESP32-based sensor data acquisition**: Collecting sensor data from various devices.
- **Raspberry Pi with RAID 10 storage**: Configuring a Raspberry Pi server with RAID 10 for performance and data redundancy.
- **IoT integration**: Enabling data transmission from the ESP32 to the Raspberry Pi over Wi-Fi.
- **Web-based dashboard**: Developing a user-friendly interface for remote data monitoring.

This project will provide a local, secure, and scalable storage solution for IoT sensors, ensuring data integrity and ease of management.

ENGINEERING PROBLEM STATEMENT

In the current digital age, the management, storage, and security of data present significant challenges for individuals and organizations. Although cloud storage has gained popularity due to its scalability, accessibility, and convenience, it introduces substantial drawbacks that can hinder performance, security, and cost-effectiveness, especially for sensitive data.

Key Issues with Cloud Storage:

1. **Security and Privacy Concerns:**
    - **Data Breaches:** Storing sensitive data on third-party cloud servers increases the risk of breaches and cyberattacks, exposing organizations to potential data theft.
    - **Compliance Issues:** Adhering to data protection regulations (such as GDPR or HIPAA) is complex when relying on cloud providers, as ensuring compliance may be outside of the organization’s control.
    - **Lack of Control:** Cloud providers control data storage policies and security measures, limiting users' ability to manage and protect their own data.
2. **Absence of a robust solution for Sensor Data Management:**
    - Despite the availability of cloud-based storage systems, there is a lack of a comprehensive and robust solution for managing and storing large volumes of sensor data locally. Storage solutions often fail to meet the performance, security, and redundancy requirements for sensor data storage. Real-time data collection, management, and local storage for IoT applications demand a more effective and reliable approach, one that offers high data availability and fault tolerance without the vulnerabilities associated with cloud-based systems.
3. **Recurring Costs:**
    - **Subscription Fees:** Cloud storage services often involve recurring subscription costs that increase as sensor data storage needs grow, making long-term use financially unsustainable, especially for large volumes of data.
    - **Cost Uncertainty:** Variable fees for data transfer, API requests, and additional cloud services can make overall costs unpredictable, complicating budgeting for organizations managing continuous sensor data.

Given these concerns, there is a growing need for a local data storage solution that offers better performance, security, and cost-efficiency. This project proposes transmitting real-time sensor data to a Raspberry Pi server, with data stored on USB drives in a RAID 10 configuration. This system eliminates cloud dependency, ensuring security, redundancy.

UNDERSTANDING RAID SETUP

RAID (Redundant Array of Independent Disks) is a technology that combines multiple physical disk drives into a single logical unit for data redundancy, performance improvement, or both. The key concept behind RAID is to distribute data across several disks to improve performance and provide fault tolerance. RAID levels use various methods like striping, mirroring, and parity to achieve these goals.

Standard Types of RAID:

1. RAID 0 (Striping)

- **Configuration:** Data is split into blocks and each block is written to a separate disk.
- **Advantages:** Improved read and write performance because data is read/written in parallel across multiple disks.
- **Disadvantages:** No redundancy. If one disk fails, all data is lost.
- **Use Case:** Suitable for non-critical systems where performance is more important than data redundancy, such as video editing or gaming.

2. RAID 1 (Mirroring)

- **Configuration:** Identical copies of data are stored on two or more disks.
- **Advantages**: High redundancy and data protection. If one disk fails, the data is still accessible from the other disk.
- **Disadvantages:** Storage capacity is halved as data is duplicated on each disk.
- **Use Case**: Ideal for critical systems where data availability and integrity are crucial, such as database servers and transaction systems.

3. RAID 2 (Bit-level Striping with Hamming Code)

- **Configuration:** Data is striped at the bit level across multiple disks, with error correction using Hamming code.
- **Advantages:** Provides error correction and fault tolerance.
- **Disadvantages:** Requires a large number of disks and is complex to implement. Not commonly used in modern systems.
- **Use Case:** Rarely used due to its complexity and the availability of more efficient RAID levels.

4. RAID 3 (Byte-level Striping with Dedicated Parity)

- **Configuration:** Data is striped at the byte level across multiple disks with a dedicated parity disk for error checking.
- **Advantages:** Improved performance over a single disk with some fault tolerance.
- Disadvantages: The dedicated parity disk can become a bottleneck. If the parity disk

fails, redundancy is lost until it is rebuilt.

- **Use Case:** Suitable for applications requiring high throughput for large files, such as video streaming.

5. RAID 4 (Block-level Striping with Dedicated Parity)

- **Configuration:** Similar to RAID 3 but with block-level striping and a dedicated parity disk.
- **Advantages:** Allows for larger block sizes and improved read performance compared to RAID 3.
- **Disadvantages:** The dedicated parity disk can still be a bottleneck, particularly during write operations.
- **Use Case:** Used in environments where read performance is critical and data redundancy is required, such as data warehouses.

6. RAID 5 (Block-level Striping with Distributed Parity)

- **Configuration:** Data and parity information are striped across all disks in the array. Parity blocks are distributed among the disks.
- **Advantages:** Provides fault tolerance and improved read performance. Only one additional disk is required for parity.
- **Disadvantages:** Write performance can be slower due to parity calculation. If a disk fails, rebuild times can be significant.
- **Use Case:** Commonly used in enterprise environments where a balance between performance, redundancy, and storage capacity is needed, such as file and application servers.

Nested RAID: RAID 10 (1+0):

RAID 10, also known as RAID 1+0, combines the features of RAID 1 and RAID 0 to provide both redundancy and improved performance.

RAID 10 Configuration:

- **Mirroring and Striping**: Data is first mirrored, and then the mirrors are striped.
- **Structure:** A RAID 10 array requires a minimum of four disks. The array is built by creating mirrored pairs (RAID 1), and then data is striped across these pairs (RAID 0).

Advantages:

- **High Performance:** Offers high read and write speeds due to striping.
- **High Redundancy:** Provides excellent fault tolerance. If a single disk in a mirrored pair fails, the array can continue to operate.
- **Fast Rebuild Times:** Only the failed disk's mirrored data needs to be copied during a rebuild, making the process faster than RAID 5.

Disadvantages:

- **Cost:** Requires double the number of disks for redundancy, which can be expensive.
- **Storage Efficiency:** Only 50% of the total disk capacity is available for data storage due to mirroring.

Use Case:

- **Ideal Scenarios:** RAID 10 is suitable for environments requiring high performance and redundancy, such as high-transaction databases, email servers, and applications demanding high availability.

For our project, implementing RAID 10 offers the best combination of performance, redundancy, and reliability. This RAID configuration ensures that data is protected against disk failures while providing fast read and write operations, making it an ideal choice for a robust and efficient IoT-based NAS system. By leveraging RAID 10, the NAS system can maintain high data availability and integrity, essential for critical storage applications.

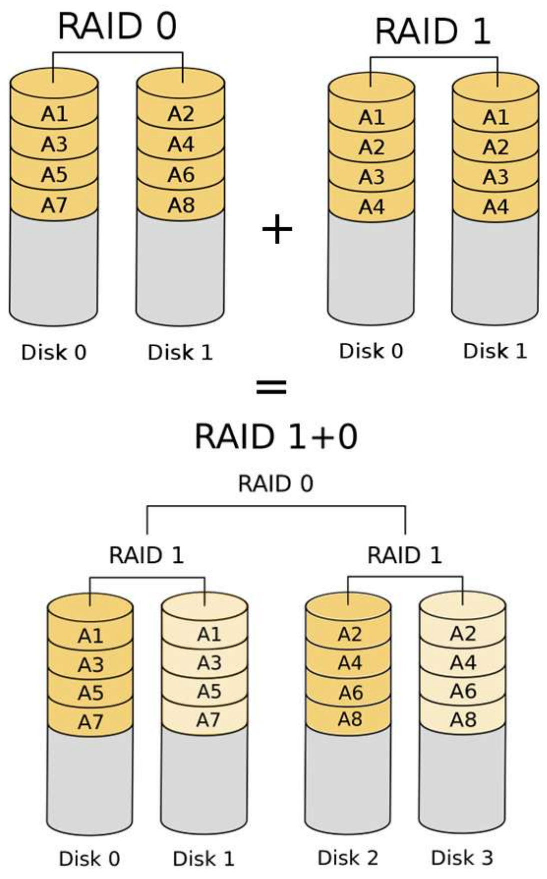


Fig1. RAID10

PROPOSED SOLUTION: IoT-Based Storage with Software RAID

To address the drawbacks of third-party cloud storage, this project proposes the development of an IoT-based local storage system integrated with software RAID 10 technology. This solution ensures a secure, high-performance, and cost-effective approach to storing real-time sensor data locally, without relying on external services.

By using ESP32 microcontrollers to wirelessly transmit sensor data to a Raspberry Pi server, this solution enables seamless, real-time data collection and storage. The Raspberry Pi uses a software RAID 10 configuration with USB storage drives, providing enhanced data redundancy, fault tolerance, and improved performance.

Advantages of IoT-Based Storage with Software RAID 10:

1. Enhanced Security and Control:
    - Local Data Storage: Data is stored locally on storage drives, ensuring full control over the data environment and security measures.
    - No External Dependencies: By eliminating reliance on third-party cloud services, data privacy and security are ensured.
    - User Control: The system is entirely user-managed, providing direct control over storage, maintenance, and access.
2. Improved Performance and Reliability:
    - RAID Redundancy: The software-based RAID 10 configuration offers both data mirroring and striping, ensuring high availability, data redundancy, and fault tolerance in case of drive failure.
    - Real-Time Data Transfer: Data from sensors is transmitted wirelessly in real time, providing immediate access for analysis and monitoring.
    - Efficient Storage: RAID 10 ensures faster data access and redundancy without the complexities and costs of hardware RAID.
3. Cost Efficiency:
    - Elimination of Cloud Subscription Costs: By using local storage, this solution eliminates the ongoing fees associated with cloud storage services.
    - Predictable Costs: Hardware investments are made upfront, and the system has minimal maintenance costs, ensuring predictable budgeting and no surprise fees.

4. Advanced Monitoring and Management:

   • IoT Integration: Telemetry data, such as temperature and other environmental factors, are fetched using IoT sensors, allowing for real-time monitoring of the system's performance.

   • Web-Based Dashboard**:** A user-friendly web dashboard enables remote monitoring of sensor data and storage health, facilitating easy management of the system.

   • Data Storage for Future Use: The telemetry data is stored in a local database, allowing for later analysis, including running AI models or other data processing tasks to extract insights.

   • Remote Alerts and Notifications: The system sends alerts for critical events, such as disk failures or unusual sensor readings, allowing users to take timely action to maintain system reliability.

By utilizing software RAID 10, this solution ensures a combination of data redundancy, fault tolerance, and high performance, all while keeping costs low and maintaining full control over the storage environment. The IoT-based system captures and stores telemetry data locally, providing secure storage for future analysis, including AI model processing, making it a scalable and reliable alternative to cloud storage for sensor data management.

TECHNOLOGIES USED

ESP32:

The ESP32 is a low-cost, low-power microcontroller with built-in Wi-Fi and Bluetooth capabilities, making it ideal for IoT applications. In this project, the ESP32 is used for collecting real-time telemetry data from environmental sensors, such as temperature, humidity, and air quality sensors. The ESP32 communicates wirelessly with the Raspberry Pi, transmitting sensor data over Wi-Fi in a secure and efficient manner. Its flexibility allows the integration of various types of sensors, enabling the system to monitor a wide range of environmental conditions. The ESP32 serves as the primary data acquisition device, capturing data from sensors and transmitting it for further processing and storage on the Raspberry Pi.

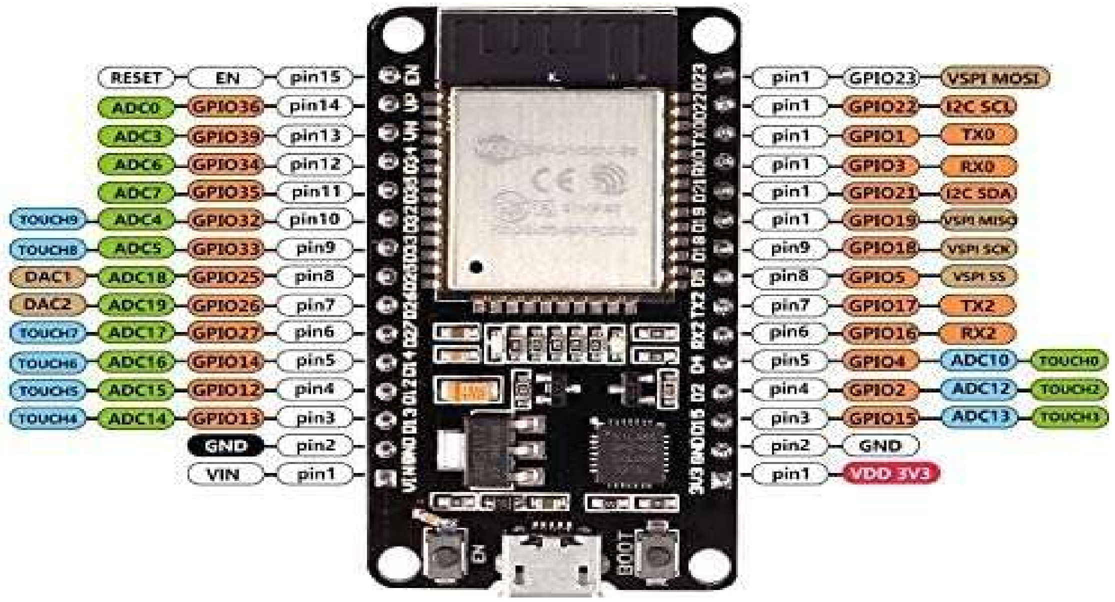
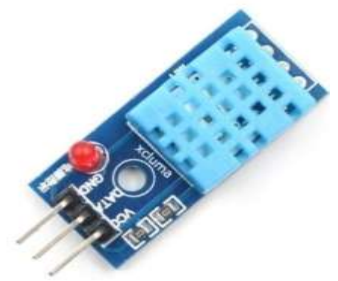


Fig2. ESP32

Sensors:

Sensors are essential components in this IoT-based system, responsible for collecting real-time environmental data such as temperature, humidity, air quality, and pressure. Common sensors used include DHT22 for temperature and humidity, MQ-series for gas detection, and BME280 for pressure and altitude. These sensors interface with the ESP32 microcontroller, which collects the data and wirelessly transmits it to the Raspberry Pi. The data is then stored in InfluxDB for future analysis and visualization in Grafana, enabling users to monitor and analyze environmental conditions in real time. These sensors provide the raw telemetry necessary for system performance, analysis, and future machine learning applications.



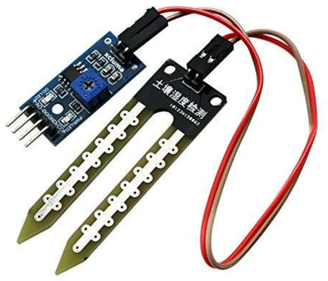


Fig3. DHT11 Sensor *Fig4. Ultrasound Sensor* *Fig5. Soil Moisture Sensor*

Raspberry Pi:

The Raspberry Pi serves as the heart of the IoT-based storage and monitoring system. A powerful yet affordable single-board computer, the Raspberry Pi offers the flexibility and capability to manage multiple IoT devices, handle sensor data processing, and control storage devices locally. In this project, the Raspberry Pi acts as the central server, interfacing with sensors, managing local storage drives, and running the necessary software for storing and processing telemetry data. By connecting to the storage drives configured in a RAID 10 setup, the Raspberry Pi ensures both redundancy and high performance. It also provides the necessary network connectivity (via Wi-Fi or Ethernet) to interface with the ESP32 sensors and enable remote data access via the web-based dashboard.

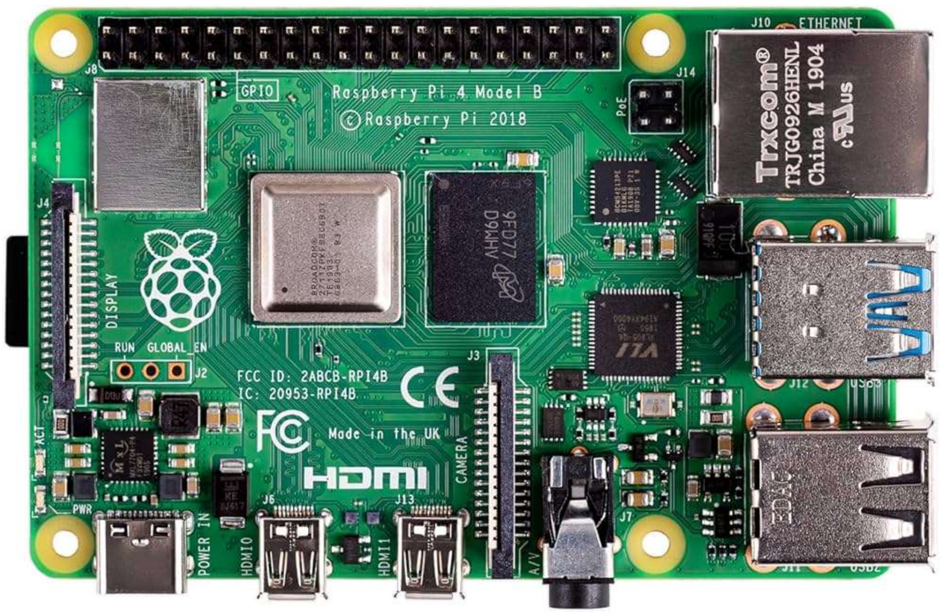


Fig6. Raspberry Pi

Storage Drives and RAID 10 Configuration:

The system uses multiple storage drives configured in RAID 10 (Redundant Array of Independent Disks, Level 10) to provide a combination of data redundancy, performance, and fault tolerance. RAID 10 is an ideal solution for systems that require high-speed data access and data protection, as it combines RAID 1 (mirroring) and RAID 0 (striping). Data is mirrored across pairs of drives, ensuring redundancy, while striping allows for faster read and write operations by splitting the data across multiple drives. This configuration allows the system to store large volumes of sensor data efficiently while safeguarding against data loss in case of drive failure. The RAID 10 software implementation is managed by tools like mdadm on Linux, which makes the setup cost-effective and flexible compared to hardware RAID solutions.

InfluxDB:

InfluxDB is an open-source, time-series database designed for efficiently storing and querying high volumes of time-stamped data. It is well-suited for handling telemetry data from IoT devices, as it is optimized for fast write operations and time-series queries. In this project, InfluxDB stores the telemetry data collected from the ESP32 sensors, including environmental parameters such as temperature, humidity, and air quality. The database allows the system to manage large amounts of sensor data over time, with efficient storage and retrieval mechanisms. InfluxDB's powerful querying capabilities enable data analysis for trend monitoring and anomaly detection. Its integration with Grafana makes it an ideal choice for real-time data visualization and monitoring.

Grafana:

Grafana is a powerful open-source data visualization and monitoring platform that integrates seamlessly with time-series databases like InfluxDB. It is used in this project to create an interactive, real-time dashboard for monitoring the sensor data stored in InfluxDB. Grafana allows users to create customizable visualizations, such as graphs, tables, and heatmaps, to track the telemetry data over time. The dashboard automatically updates as new sensor data is received, providing users with real-time insights into environmental conditions. Grafana's user-friendly interface makes it easy to filter and analyze data, view historical trends, and set up alerts based on specific conditions (e.g., temperature exceeding a threshold). This integration provides a scalable, efficient solution for monitoring and managing large sets of sensor data.

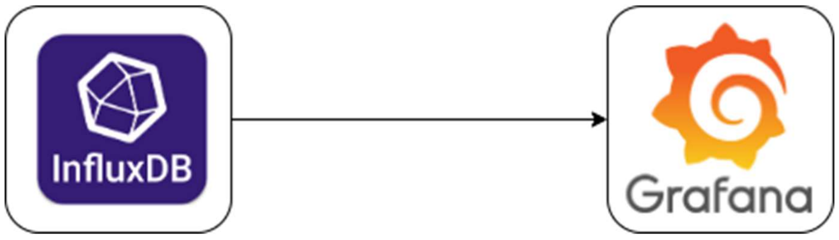


Fig7. InfluxDB to Grafana

RAID 10 Software Implementation:

RAID 10, also known as RAID 1+0, is a combination of RAID 1 (mirroring) and RAID 0 (striping) that offers both redundancy and high performance. In this project, the software implementation of RAID 10 involves configuring multiple storage drives in a way that balances fault tolerance with optimal read and write speeds. The primary advantage of RAID 10 is its ability to provide data redundancy through mirroring, meaning that data is duplicated across multiple drives. If one drive fails, the mirrored copy on the other drive ensures that no data is lost, offering high availability and resilience.

On the other hand, RAID 10 also uses striping, where data is divided into smaller chunks and spread across multiple drives. This improves read and write performance since multiple drives can work in parallel to access or store data, resulting in faster data retrieval and processing times. The combination of mirroring and striping means that RAID 10 offers a high level of fault tolerance without compromising speed, making it ideal for applications that require both reliability and performance, such as real-time sensor data collection and analysis.

In this project, the RAID 10 software implementation is managed through the Raspberry Pi, using Linux-based tools such as mdadm to configure and maintain the RAID array. The software RAID approach allows for flexibility in managing the storage drives, providing the ability to configure, monitor, and repair the RAID array without needing specialized hardware. Additionally, software RAID is cost-effective since it does not require dedicated RAID hardware controllers. The RAID 10 setup ensures that sensor data collected from the ESP32 is stored securely and efficiently, allowing for both fault tolerance and high-speed access for future analysis, including AI and machine learning tasks.

Web-Based Dashboard:

The web-based dashboard provides an intuitive and interactive interface for users to monitor and manage the system. Powered by Grafana, this dashboard allows users to visualize sensor data in real time through customizable graphs, charts, and tables. The dashboard is hosted on the Raspberry Pi, accessible remotely via a web browser, and continuously updated with the latest sensor readings stored in InfluxDB. It enables users to filter data based on specific criteria, such as sensor type or time range, and view historical trends or alerts triggered by predefined conditions. The web dashboard is designed to provide easy-to-understand insights into the health of the system and the environment, making it an essential tool for real-time monitoring and decision-making.

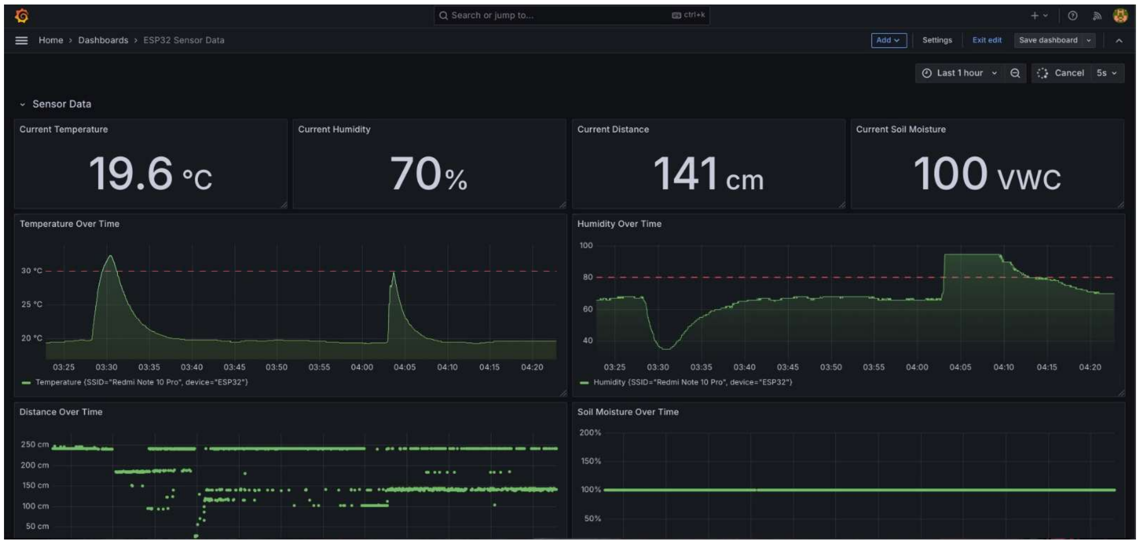


Fig8. Grafana Dashboard

MDADM:

mdadm (short for Multiple Device Admin) is a powerful Linux utility used to create, manage, and monitor software RAID (Redundant Array of Independent Disks) arrays. Unlike hardware RAID, which relies on dedicated controllers, mdadm enables RAID functionality entirely through software, making it a cost-effective and flexible solution for building high-performance and fault-tolerant storage systems. It supports various RAID levels, including RAID 0, 1, 5, 6, 10, and more, offering a balance of redundancy, speed, and storage efficiency. With mdadm, users can easily configure RAID arrays, add or remove disks, monitor array health, and set up automated notifications in the event of disk failures or array degradation. Its simplicity, combined with robust features, makes mdadm a popular choice for enterprises, data centers, and personal servers looking to ensure data integrity and availability without the need for specialized hardware.

MDADM WORKING

Step 1: Disk Identification and Preparation

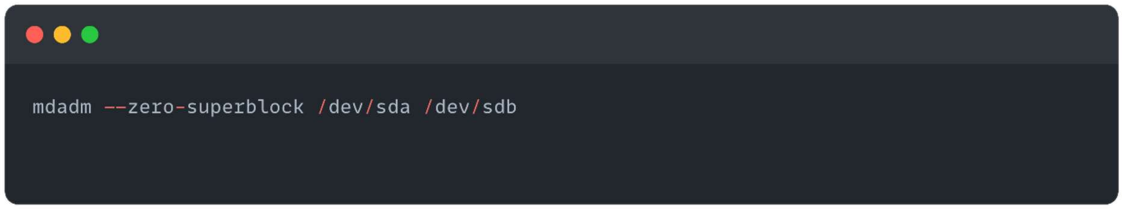
Before creating a RAID 10 array, mdadm must identify and prepare the physical storage devices. This step ensures that all devices intended for the RAID array are available, functional, and ready to be used. When you specify the devices (/dev/sda, /dev/sdb, etc.), mdadm performs the following actions in the background:

Device Enumeration: The command checks the /dev directory to list all available block devices. It ensures that the devices you specified exist and are recognized by the operating system.
Partition Table Check: mdadm verifies whether the specified devices have existing partition tables or filesystems. If a partition table or filesystem exists, it warns the user to prevent accidental data loss.

Device Status Check: The command also checks if the devices are already in use by another RAID array or mounted as regular block devices.

Clearing Superblocks: If a device was previously part of another RAID array, it might contain old RAID metadata. mdadm can clear the old superblock using the --zero-superblock option. This step is necessary to avoid conflicts with the new array.

Example Command:



```
mdadm --zero-superblock /dev/sda /dev/sdb
```

Disk Preparation: If the disks are new and unpartitioned, you can proceed directly. If they contain data, they may need to be wiped or partitioned depending on the intended usage.
Device Initialization: Once the checks are complete, the devices are ready for the RAID array creation.

Step 2: RAID 10 Array Creation

RAID 10 combines RAID 1 (mirroring) and RAID 0 (striping). When you execute mdadm --create to create a RAID 10 array, it performs several actions behind the scenes:

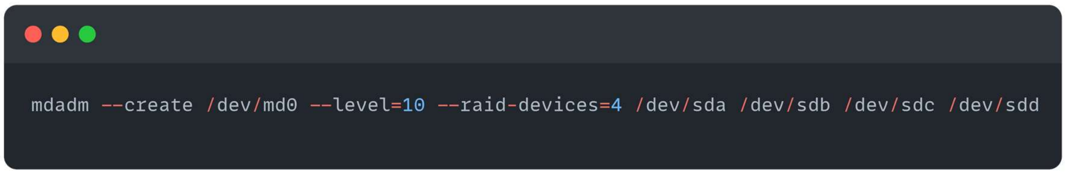
Disk Verification: mdadm verifies the availability of all specified devices and checks for conflicts, such as if the disks are in use or have existing data.
Kernel Interaction: mdadm communicates with the Linux kernel's MD (Multiple Device) driver using IOCTL system calls to instruct the kernel to create a new RAID device. The key IOCTL command used is CREATE_ARRAY, which initiates the array creation.

RAID Level Setup: The command specifies the RAID level (10) and the number of devices involved. The kernel configures the MD driver to implement this specific RAID configuration.

Superblock Initialization: mdadm writes metadata (the superblock) to each disk in the array. The superblock includes:

- Array UUID (unique identifier)
- RAID level (10)
- Number of devices and their roles
- Data layout, chunk size, and other configuration parameters

Device Node Creation: The kernel creates a new block device node in /dev, typically named /dev/md0, which represents the RAID array. This device can be used like a regular block device.
Example Command:



```
mdadm --create /dev/md0 --level=10 --raid-devices=4 /dev/sda /dev/sdb /dev/sdc /dev/sdd
```

Step 3: Data Management (Striping and Mirroring)

In RAID 10, data is distributed across mirrored pairs of disks, providing both performance and redundancy. Here's how data management works:

Data Chunking: Incoming data is divided into chunks based on the specified chunk size (e.g., 64 KB).

Mirroring (RAID 1): Each chunk is duplicated across a pair of disks. For example, if you have four disks:

- Chunk 1 is written to Disk 1 and Disk 2 (mirrored pair).
- Chunk 2 is written to Disk 3 and Disk 4 (another mirrored pair).

Striping (RAID 0): After the data is mirrored, it is striped across the mirrored pairs for improved performance. The MD driver writes data to each mirrored pair in a round-robin fashion.

Data Layout Example:

Chunk	**Disk 1**	**Disk 2**	**Disk 3**	**Disk 4**
Chunk1	Data A	Data A		
Chunk2			Data B	Data B
Chunk3	Data C	Data C		
Chunk4			Data D	Data D

Performance and Redundancy: Striping improves performance by distributing I/O across multiple disks, while mirroring ensures redundancy.

Step 4: Handling I/O Requests (Reads and Writes)

Once the RAID 10 array is active, it handles I/O requests differently depending on the operation:
Read Operations: The MD driver distributes read requests across all available disks to maximize performance. Since data is mirrored, it can read from either disk in a mirrored pair, balancing the load.

Read Optimization: In RAID 10, the MD driver often chooses the least busy disk in a mirrored pair for read operations, improving overall throughput.
Example:

- If a read request is for Chunk 1, it can be fulfilled by either Disk 1 or Disk 2.
- The kernel selects the disk with the lower current I/O load.

Write Operations: Write operations are duplicated to both disks in a mirrored pair.
Write Process:

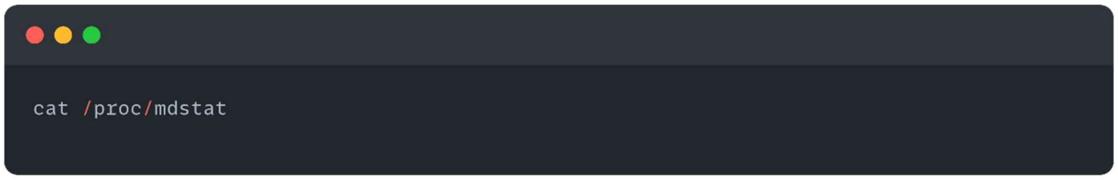
1. The MD driver writes the chunk to both disks in the pair.
2. It stripes the next chunk across the next mirrored pair.

Write Penalty: Because each write is duplicated, RAID 10 write speeds are slightly slower than read speeds but still faster than RAID 1 alone.

Step 5: Monitoring and Maintenance

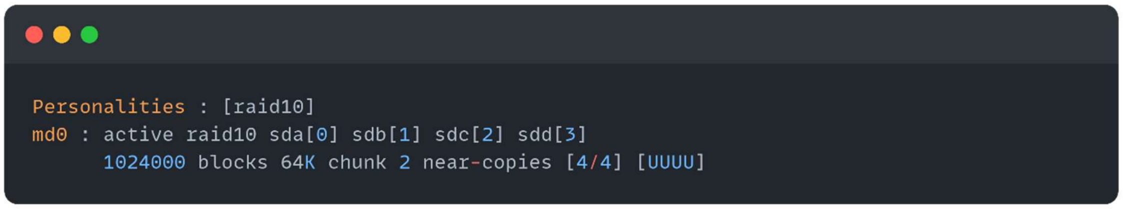
After the RAID 10 array is created and running, mdadm continuously monitors the health and performance of the array.

Status Check: You can check the status of the array using:



```
cat /proc/mdstat
```

Example Output:



```
Personalities : [raid10]
md0 : active raid10 sda[0] sdb[1] sdc[2] sdd[3]
      1024000 blocks 64K chunk 2 near-copies [4/4] [UUUU]
```

Disk Health Monitoring: The MD driver periodically checks the health of each disk. If a disk becomes unresponsive or fails, it is marked as faulty.

Step 6: Rebuilding and Resyncing

When a disk in the RAID 10 array fails, mdadm can replace it and rebuild the array to restore redundancy.
Failure Detection: The MD driver detects the failed disk and marks the array as degraded.

Example Degraded Array:

```
md0 : active raid10 sda[0] sdb[1] sdc[2] [U_UU]
```

Adding a Replacement Disk: To replace a failed disk, first remove the faulty disk from the array:

```
mdadm --remove /dev/md0 /dev/sda
```

Add the new disk:

```
mdadm --add /dev/md0 /dev/sde
```

Rebuild Process:

1. The MD driver identifies the healthy disk in the mirrored pair.
2. It synchronizes data from the healthy disk to the replacement disk.
3. During the rebuild, the array remains functional, but performance may be slightly degraded.

Monitoring Rebuild Status:

```
cat /proc/mdstat
```

Rebuild Completion: Once the rebuild is complete, the array is no longer degraded, and redundancy is fully restored.

IMPLEMENTATION

The implementation of the IoT-based real-time data acquisition and storage system with a software-based RAID 10 setup involves several steps to ensure proper integration of hardware, software, and the web-based dashboard. Below are the detailed steps for the successful implementation of this project:

1. Hardware Setup

- ESP32 Sensor Nodes:
  - Set up an ESP32 microcontrollers, each connected to a specific sensor (e.g., temperature, humidity, or pressure sensors).
  - Configure the ESP32 boards to send sensor over Wi-Fi.
- Raspberry Pi Server:
  - Install the necessary software (e.g., Raspbian OS) on the Raspberry Pi to act as the central server for receiving and storing sensor data.
  - Ensure the Raspberry Pi is connected to the network for communication with the ESP32 nodes.
- Storage Configuration (RAID 10):
  - Connect multiple USB storage drives to the Raspberry Pi to create a local storage array.
  - Set up software-based RAID 10 on the Raspberry Pi using a tool like mdadm to configure the RAID array, ensuring redundancy and fault tolerance.

2. Data Collection and Transmission

- Sensor Data Acquisition:
  - Program the ESP32 microcontrollers to collect data from the connected sensors and convert it into a suitable format (JSON).
  - Set up the ESP32 to transmit this data wirelessly to the Raspberry Pi using a secure Wi-Fi connection (HTTP protocol).
- Data Transmission to Raspberry Pi:
  - Configure the Raspberry Pi to receive the data from multiple ESP32 nodes.
  - Set up the communication protocol (e.g., MQTT or HTTP) on the Raspberry Pi to listen for incoming data from the ESP32 devices.

3. Data Storage Configuration

- RAID 10 Setup:
  - On the Raspberry Pi, configure the RAID 10 setup using software tools like mdadm. This configuration will mirror the data across multiple drives for

    - redundancy while also improving read/write performance.
    - Ensure the RAID system is initialized and the drives are formatted for use.

- Local Data Storage:
    - Program the Raspberry Pi to store incoming sensor data on the RAID 10 storage array. The data should be organized into folders by sensor type and time-stamped for easy retrieval.
    - Implement data management scripts to handle storage overflow and regular backups if necessary.

4. Web-Based Dashboard Development

- Database Integration:
    - Set up InfluxDB on the Raspberry Pi to store real-time sensor data collected from the ESP32. Grafana will be configured to connect to InfluxDB to fetch the latest data stored in the database.
    - InfluxDB's time-series capabilities ensure efficient storage and retrieval of telemetry data, such as temperature, humidity, and other sensor readings, which Grafana will then query to display on the dashboard.
    - Configure data queries within Grafana to pull updated sensor information at regular intervals, ensuring that the data visualized in Grafana is current and accurate.
- Dashboard Implementation:
    - Use Grafana's intuitive, user-friendly interface to create customizable dashboards that visualize real-time sensor data through graphs, tables, and charts.
    - The dashboard will automatically update as new data is fetched from InfluxDB, providing dynamic visualizations of the telemetry data.
    - Grafana will include features for filtering data based on specific time frames, sensor types, or other criteria, allowing users to interact with historical data and track trends over time for more detailed analysis.

5. Testing and Validation

- Test Data Transmission:
    - Test the wireless data transmission from the ESP32 nodes to the Raspberry Pi by ensuring that the sensors send data at regular intervals.
    - Verify that the Raspberry Pi receives and stores the data correctly.
- RAID and Storage Validation:
    - Test the RAID 10 configuration to ensure data redundancy and fault tolerance by simulating drive failures and verifying that data remains accessible.
- Real-Time Monitoring on the Dashboard:

- Validate the real-time update functionality of the web dashboard by monitoring data changes as new sensor readings come in.

6. Optimization and Final Adjustments

- Performance Tuning:
  - Optimize the software RAID 10 setup to ensure optimal read and write speeds for data storage and retrieval.
  - Fine-tune the wireless communication between the ESP32 nodes and the Raspberry Pi to reduce transmission latency.
- Security Implementation:
  - Implement encryption and secure communication (e.g., HTTPS or MQTT over SSL/TLS) to protect data during transmission.
  - Set up user authentication on the web dashboard to ensure that only authorized users can access the data.

7. Documentation and Deployment

- System Documentation:
  - Document the hardware setup, software configurations, and steps to operate the system. Include troubleshooting tips for common issues.
- Deployment:
  - Deploy the system and monitor the performance of the sensors, Raspberry Pi, and storage system.
  - Ensure the system operates seamlessly, with data being stored and displayed in real-time on the dashboard.

By following these implementation steps, this system will provide a fully functional, secure, and cost-effective solution for real-time data collection, storage, and visualization without relying on third-party cloud services**.**

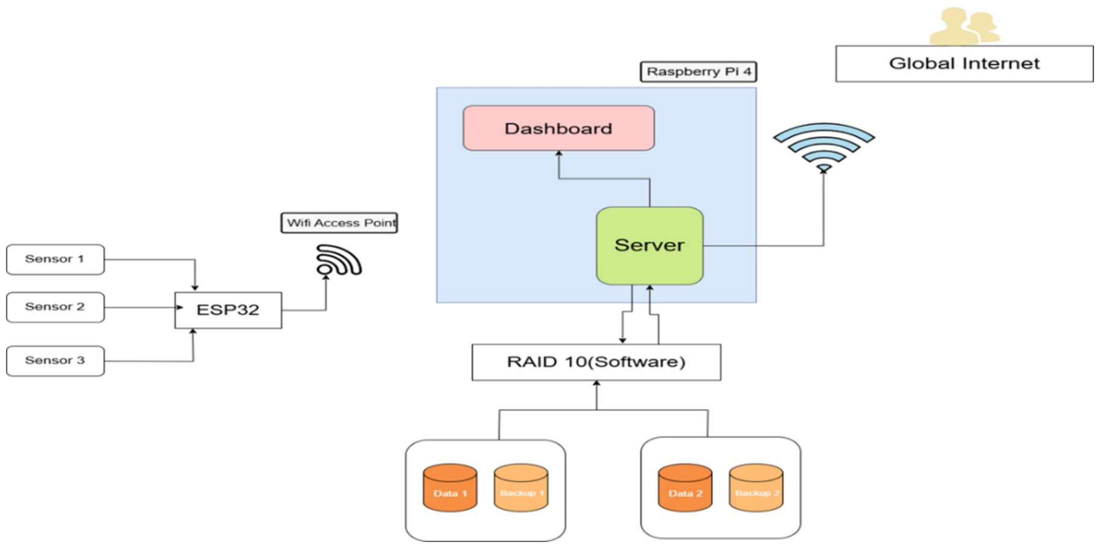


Fig9. System Architecture

CONCLUSION

In conclusion, the implementation of an IoT-based data acquisition and storage system using a Raspberry Pi server and software-based RAID 10 offers a reliable, secure, and cost-effective solution to the growing demands for real-time data management. This project addresses key concerns such as data privacy, security, and the high costs associated with cloud storage services by providing a self-contained, on-premise system for storing and visualizing sensor data.

By leveraging the ESP32 microcontrollers to wirelessly transmit sensor data to the Raspberry Pi, the system ensures real-time data collection and seamless integration with local storage. The software-based RAID 10 configuration guarantees data redundancy and fault tolerance, protecting against potential drive failures while maintaining high performance. This approach eliminates the need for expensive physical RAID systems, offering a flexible and scalable alternative.

The inclusion of IoT sensors allows for proactive monitoring of environmental factors, ensuring that the system's health and data integrity can be tracked in real-time. The web-based dashboard provides a user-friendly interface for monitoring data and managing the system remotely, empowering users with real-time insights and control over the system's operations.

Security is a top priority, with robust encryption and authentication mechanisms safeguarding sensitive data from unauthorized access. The use of local storage ensures that data remains under the user's control, minimizing the risks associated with third-party cloud services.

Through extensive testing, the system has demonstrated reliable performance and high availability, meeting the demands of modern data management. The solution is highly adaptable and can be deployed across various industries requiring secure, real-time data acquisition and storage.

Looking ahead, the system has the potential for further optimization and integration of advanced features. By staying aligned with emerging technologies, this IoT-based data acquisition and storage system can evolve into an even more versatile and resilient solution, addressing future challenges in data management and security.

REFERENCES

1. Padala, H.V.R., Vurukonda, N., Mandhala, V.N., Valluru, D., Tangirala, N.S.R. and Lakshmi Manisha, J., 2021. Private cloud for data storing and maintain integrity using Raspberry Pi. In *Machine Intelligence and Soft Computing: Proceedings of ICMISC 2020* (pp. 335-350). Singapore: Springer Singapore.
2. Medina-Santiago, A., Azucena, A.D.P., Gómez-Zea, J.M., Jesús-Magaña, J.A., de la Luz Valdez-Ramos, M., Sosa-Silva, E. and Falcon-Perez, F., 2019. Adaptive model IoT for monitoring in data centers. *IEEE Access*, *8*, pp.5622-5634.
3. Karthikeyan, S., Raj, R.A., Cruz, M.V., Chen, L., Vishal, J.A. and Rohith, V.S., 2023. A systematic analysis on raspberry pi prototyping: Uses, challenges, benefits, and drawbacks. *IEEE Internet of Things Journal*, *10*(16), pp.14397-14417.
4. Chithambaramani, R., Sankar, M., Sivaprakash, P., Marichamy, D. and Yazhinian, S., 2023, November. Storage Network Attached Using Raspberry PI. In *2023 Computation, Automation and Networking (ICSCAN)* (pp. 1-6). IEEE.
5. Linux man-pages project, "mdadm(8) – manage Linux MD arrays," *Linux Man Pages, 2023*. [Online]. Available: https://man7.org/linux/man-pages/man8/mdadm.8.html.
6. Raspberry Pi Foundation, "Raspberry Pi," [Online]. Available: https://www.raspberrypi.com/. [Accessed: Dec. 6, 2024].
7. Freepik, "Freepik: Free Graphic Resources," [Online]. Available: https://www.freepik.com/. [Accessed: Dec. 6, 2024].