Technical Document

Pre-failure alert generation for equipment & cobble reduction based on data analytics and video analytics at BRM

Document version: 0.1.0

Department of Electrical Engineering and Computer Science,

IIT Bhilai, GEC Campus, Sejbahar, Raipur - 492015, (C.G.)

Technical Document for

Pre-failure alert generation for equipment & cobble reduction based on data analytics and video analytics at BRM

Document version: 0.1.0

October – 2022

Reviewed By

____

Name of person

Designation, Department, Organisation

Approved By

____

Name of person

Designation, Department, Organisation

Department of Electrical Engineering and Computer Science,

IIT Bhilai, GEC Campus, Sejbahar, Raipur - 492015, (C.G.)

Revision Records

Revision Records		
Version number	**Description**	**Document release date**
0.1.0	Initial release	October - 2022

Preface

The Technical document provides the basic information regarding the APIs, SDKs, system design and other logic to the developers and engineers in the project team.

Terms of Use and Disclaimers

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Scope of the document

1.2 Document Overview

2. System Design and Overview

2.1 Server system design and their standards

2.2 Server system connections and interaction between other systems

3. Storage Structure

3.1 File system design

We have decided to use ZFS as our file system, also I have designed the storage hierarchy of our feeds. I'll be mentioning how file system, and other storage management logics will be working.

Partitioning of Drives: For production server we will setup the servers and drives on own. Though there will be a script/program which will help us to automate the server setup for the first time use.

Programs like fdisk, cfdisk etc. but I'll be using sfdisk as it offer a non-interactive methods to partition our drive. Like we can first manually partition and structure our drive at first and then we can make a backup copy of that structure and later can be applied on the same.

I have tested that on our current server at IITB Electrical Lab on the HDD i.e. TOSHIBA MG04ACA200E (FP5B). Below are the current structure which is in raidz1.

```
label
: gpt
        label-id: E3B7FE9A-1618-4E8B-9EC7-4CA0340043DF
        device: /dev/sdb
        unit: sectors
        first-lba: 34
        last-lba: 3907029134


        /dev/sdb1 : start=        2048, size=   195313664, type=0FC63DAF-8483-4772-
        8E79-3D69D8477DE4, uuid=010F6E0C-BE91-42D9-9F1B-6885ACB3387F
        /dev/sdb2 : start=   195315712, size=   195313664, type=0FC63DAF-8483-4772-
        8E79-3D69D8477DE4, uuid=C183F619-DBD8-4728-82B8-BD369B5458D4
        /dev/sdb3 : start=   390629376, size=   195313664, type=0FC63DAF-8483-4772-
        8E79-3D69D8477DE4, uuid=DFFCD8B4-7800-4784-9236-CFC86C7F0C0F
        /dev/sdb4 : start=   585943040, size=   195313664, type=0FC63DAF-8483-4772-
        8E79-3D69D8477DE4, uuid=B598E863-37D6-4D27-9299-D9D5B636A410
        /dev/sdb5 : start=   781256704, size=   195313664, type=0FC63DAF-8483-4772-
        8E79-3D69D8477DE4, uuid=8E9C1E9E-FFE9-49F3-85CE-1AFEFF13EEC3
        /dev/sdb6 : start=   976570368, size=   195313664, type=0FC63DAF-8483-4772-
        8E79-3D69D8477DE4, uuid=851EF8D8-CE80-49C7-9750-40C3B9996FFD
```

Using the command:

```
sfdisk -d /dev/sdb >
IITB_Sample_Server_partition_with_raidz1_layout.sfd_layout
```

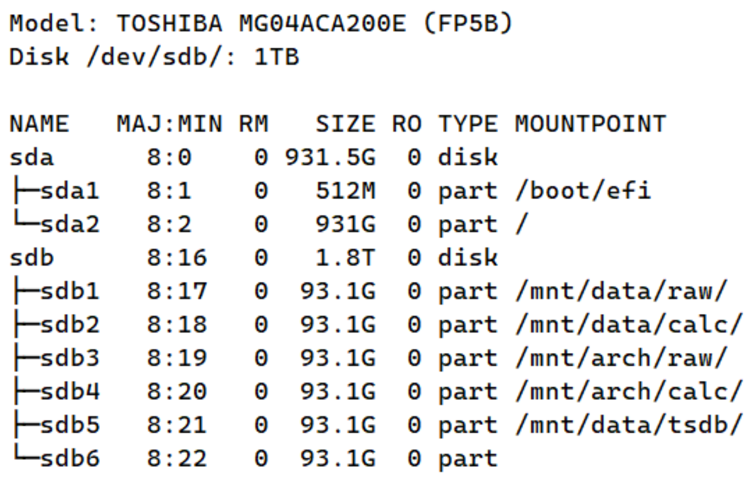
And can be applied using:

```
sfdisk -d /dev/sdb <
IITB_Sample_Server_partition_with_raidz1_layout.sfd_layout
```

As sfdisk is a part of util-linux just like fdisk, so availability should be the same.

Mount points:

Sample partition structure below, parted using sfdisk.

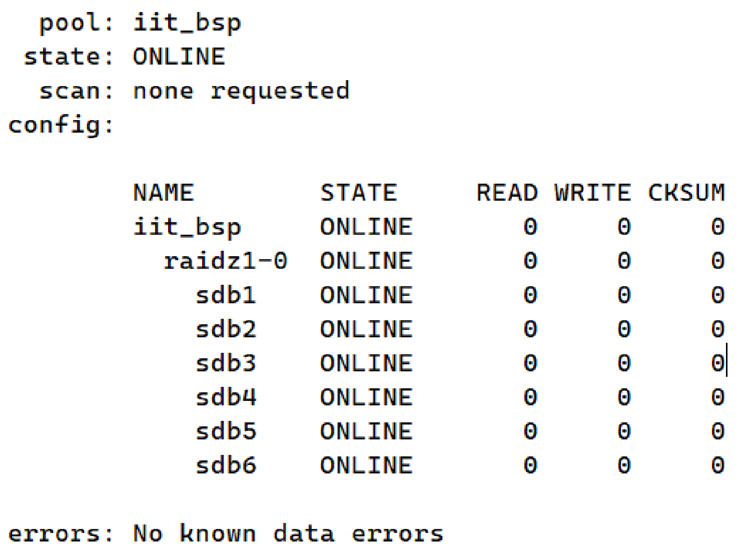


```
Model: TOSHIBA MG04ACA200E (FP5B)
Disk /dev/sdb/: 1TB

NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda       8:0    0 931.5G  0 disk
├─sda1    8:1    0   512M  0 part /boot/efi
└─sda2    8:2    0   931G  0 part /
sdb       8:16   0   1.8T  0 disk
├─sdb1    8:17   0  93.1G  0 part /mnt/data/raw/
├─sdb2    8:18   0  93.1G  0 part /mnt/data/calc/
├─sdb3    8:19   0  93.1G  0 part /mnt/arch/raw/
├─sdb4    8:20   0  93.1G  0 part /mnt/arch/calc/
├─sdb5    8:21   0  93.1G  0 part /mnt/data/tsdb/
└─sdb6    8:22   0  93.1G  0 part
```

Figure 1 Sample Partition Structure

Zpool status, formatted using zpool tool.



```
  pool: iit_bsp
 state: ONLINE
  scan: none requested
config:

        NAME          STATE     READ WRITE CKSUM
        iit_bsp       ONLINE       0     0     0
          raidz1-0    ONLINE       0     0     0
            sdb1      ONLINE       0     0     0
            sdb2      ONLINE       0     0     0
            sdb3      ONLINE       0     0     0
            sdb4      ONLINE       0     0     0
            sdb5      ONLINE       0     0     0
            sdb6      ONLINE       0     0     0

errors: No known data errors
```

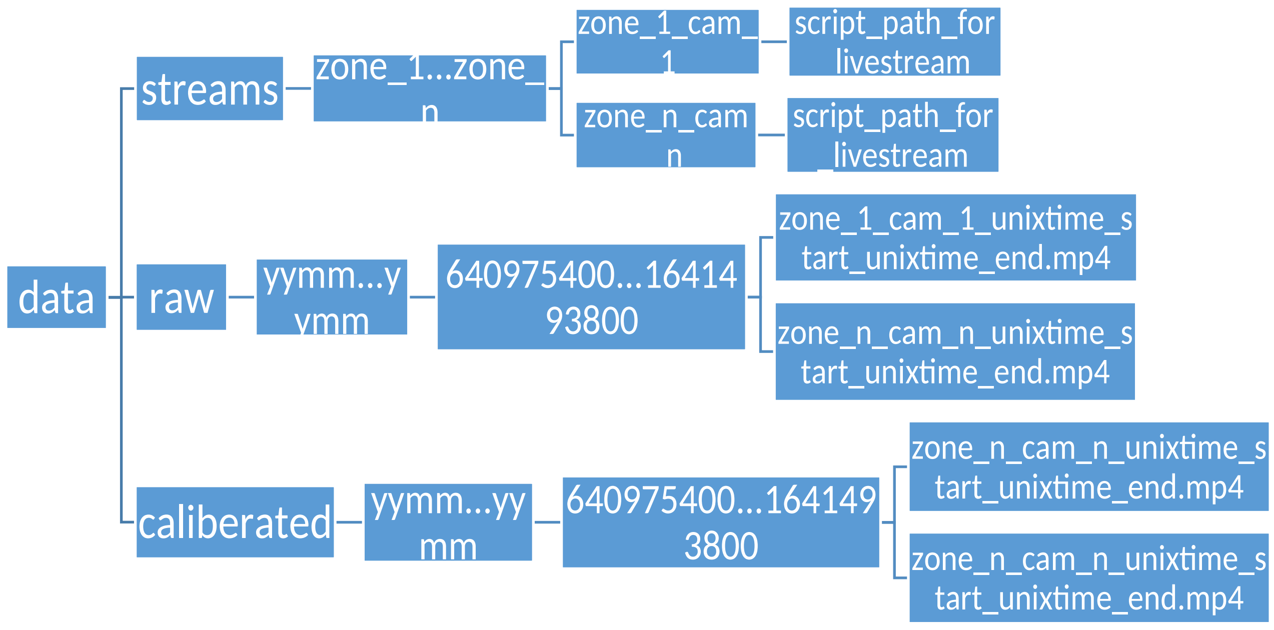
Figure 2 Zpool status

3.2 Raid and raid controller standards and design logic

3.3 Data storage design

All the data for video streams will be mounted under /data. We will keep raw feeds for about 1 week (This may change in future depending on us) along with the processed (Calibrated) data. Then after we will keep it under /archives (maybe even in compressed format using ZSTD).

Note: This hierarchy design is only for video feeds not for data which is stored in our time series database. I may well make another partition and mount there not with /data, again to make data retrieval faster and keep TMDB safe in case if /data may go corrupt.



data
streams
zone_1...zone_n
zone_1_cam_1
script_path_for livestream
zone_n_cam n
script_path_for _livestream
raw
yymm...yymm
640975400...1641493800
zone_1_cam_1_unixtime_start_unixtime_end.mp4
zone_n_cam_n_unixtime_start_unixtime_end.mp4
caliberated
yymm...yymm
640975400...1641493800
zone_n_cam_n_unixtime_start_unixtime_end.mp4
zone_n_cam_n_unixtime_start_unixtime_end.mp4

Figure 3/data partition hierarchy

I have divided the streams into zones and will categorise the respective cameras along with their zones. And as this path will be a live stream so there will be a script or a binary which will access the live feed from that camera with GigE/RTSP protocol in a GUI window with the IP:PORT rather than having a video file over there.

We will cut the video feeds from the live streams continuously like in an interval of 5 min and will be stored under /data/raw/unixdatetime/zone_n_cam_n_unixtimestart_unixtimeend.mp4 for both the raw data (feeds) as well as for calibrated data but under /data/calibrated/*.

```
data
├── caliberated
|   ├── 202209
|   |   ├── 1641061800
|   |   |   ├── blast_furnance_zone_infrared_cam_1641062400_1641062700.mp4
|   |   └── 640975400
|   |       ├── blast_furnance_zone_infrared_cam_1640976000_1640976300.mp4
|   |       └── thermal_zone_monochrome_cam_1640976000_1640976300.mp4
|   └── 202210
|       └── 640975400
|           ├── blast_furnance_zone_infrared_cam_1640976000_1640976300.mp4
|           └── thermal_zone_monochrome_cam_1640976000_1640976300.mp4
├── raw
|   ├── 202209
|   |   ├── 1641061800
|   |   |   ├── blast_furnance_zone_infrared_cam_1641062400_1641062700.mp4
|   |   |   └── thermal_zone_monochrome_cam_1641062400_1641062700.mp4
|   └── 202210
|       ├── 1641061800
|       |   ├── blast_furnance_zone_infrared_cam_1641062400_1641062700.mp4
|       |   └── thermal_zone_monochrome_cam_1641062400_1641062700.mp4
|       └── 640975400
|           ├── blast_furnance_zone_infrared_cam_1640976000_1640976300.mp4
└── streams
    ├── blast_furnance_zone
    |   ├── infrared_cam_stream
    |   └── monochrome_cam_stream
    ├── other_zones_stream
    |   ├── infrared_cam_stream
    |   └── monochrome_cam_stream
    └── thermal_zone
        ├── infrared_cam_stream
        └── monochrome_cam_stream
```

Figure 4 Sample /data hierarchy

Some points

- There are many different ways in which we can define our storage hierarchy, for now, I have done it in the date-time manner in which all videos of the day1 are stored under the day1 folder and so on for day^n and then just the video files with 5min of interval and to query the videos we can go through the DateTime folder and the name of the file. The name of the file will contain the zone and camera from which it was taken.

- Other ways can be categorizing it in zones or the camera from which was taken and then the DateTime folder or just appending that in the file. (/data/zone1...n/datetime/video_files or /data/cam1...n/datetime/video_files).

3.3.1 Archive storage design

As we will archive the raw as well as processed data there will be a separate partition for archives which is /archives. The raw data will probably be kept for 1 year at max and 2 years max for processed data (which may change in future).

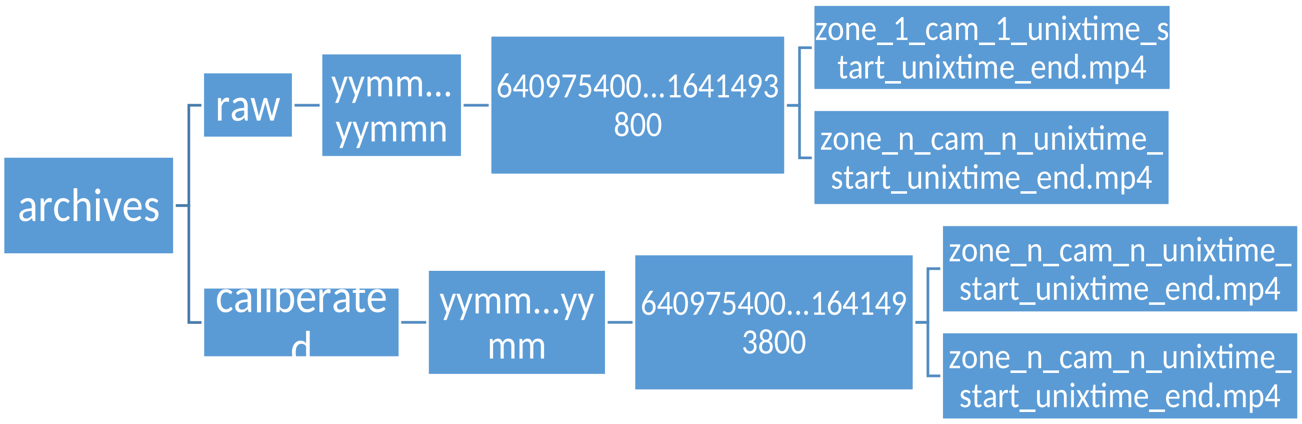
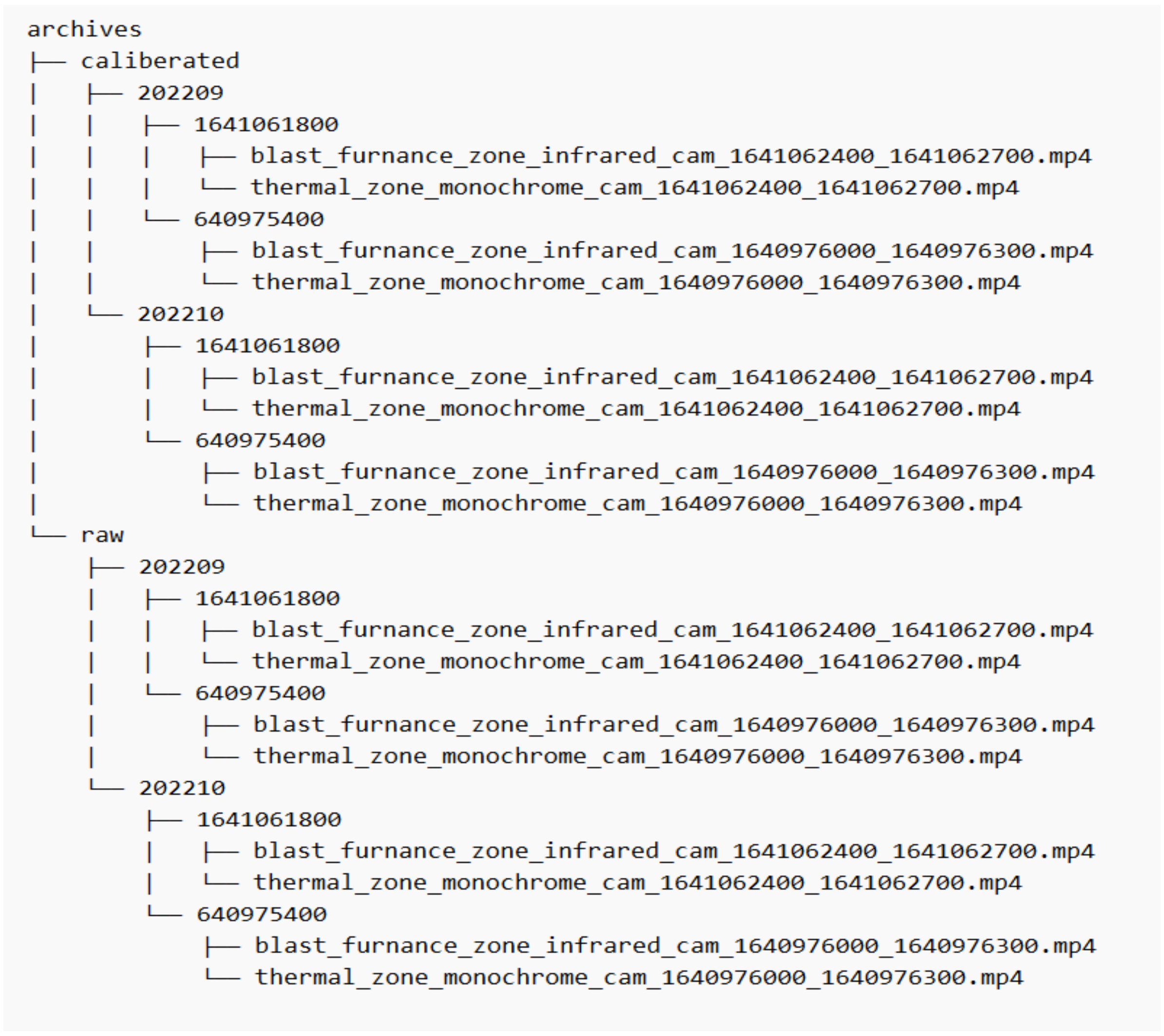

archives
raw
yymm... yymmn
640975400...1641493800
zone_1_cam_1_unixtime_start_unixtime_end.mp4
zone_n_cam_n_unixtime_start_unixtime_end.mp4
caliberated
yymm...yymm
640975400...1641493800
zone_n_cam_n_unixtime_start_unixtime_end.mp4
zone_n_cam_n_unixtime_start_unixtime_end.mp4

Figure 5 /archives partition hierarchy



```
archives
├── caliberated
|   ├── 202209
|   |   ├── 1641061800
|   |   |   ├── blast_furnance_zone_infrared_cam_1641062400_1641062700.mp4
|   |   |   └── thermal_zone_monochrome_cam_1641062400_1641062700.mp4
|   |   └── 640975400
|   |       ├── blast_furnance_zone_infrared_cam_1640976000_1640976300.mp4
|   |       └── thermal_zone_monochrome_cam_1640976000_1640976300.mp4
|   └── 202210
|       ├── 1641061800
|       |   ├── blast_furnance_zone_infrared_cam_1641062400_1641062700.mp4
|       |   └── thermal_zone_monochrome_cam_1641062400_1641062700.mp4
|       └── 640975400
|           ├── blast_furnance_zone_infrared_cam_1640976000_1640976300.mp4
|           └── thermal_zone_monochrome_cam_1640976000_1640976300.mp4
└── raw
    ├── 202209
    |   ├── 1641061800
    |   |   ├── blast_furnance_zone_infrared_cam_1641062400_1641062700.mp4
    |   |   └── thermal_zone_monochrome_cam_1641062400_1641062700.mp4
    |   └── 640975400
    |       ├── blast_furnance_zone_infrared_cam_1640976000_1640976300.mp4
    |       └── thermal_zone_monochrome_cam_1640976000_1640976300.mp4
    └── 202210
        ├── 1641061800
        |   ├── blast_furnance_zone_infrared_cam_1641062400_1641062700.mp4
        |   └── thermal_zone_monochrome_cam_1641062400_1641062700.mp4
        └── 640975400
            ├── blast_furnance_zone_infrared_cam_1640976000_1640976300.mp4
            └── thermal_zone_monochrome_cam_1640976000_1640976300.mp4
```

Figure 6 Sample /archive hierarchy

3.4 Data management logics

Storage Purging Logic

- I'm thinking to use the UNIX cron daemon for this regular purging process, as it was developed by AT&T and is part of the standard UNIX daemon so we can rely on that.

- Here is the basic cron expression:

```
# ┌───────────── minute (0 - 59)
# │ ┌───────────── hour (0 - 23)
# │ │ ┌───────────── day of the month (1 - 31)
# │ │ │ ┌───────────── month (1 - 12)
# │ │ │ │ ┌───────────── day of the week (0 - 6) (Sunday to Saturday;
# │ │ │ │ │                                   7 is also Sunday on some systems)
# │ │ │ │ │
# │ │ │ │ │
# * * * * * <command to execute>
```

- Let us take an example, we have millions of video feeds under /archive and we want to make sure the video feeds which are older than 2 years will be deleted regularly. So, we will make a script which will check whether there is any video or not if NULL or empty then return else check for videos whose DateTime is 2 years older than the current system time all this process will be run every day, so to this in cron:

```
0 0 * * * <path_to_our_deleting_script_or_binary> // Every day at 12:00 AM
Or
@daily <path_to_our_deleting_script_or_binary> // Equivalent to above expression
```

- With cron daemon, we can have many combinations for our needs.
- crond also offers us to allow and deny the cron service for the users under /etc/cron.allow and /etc/cron.deny.
- And the same goes for /data, after 1 week or whatever time that we will define the raw and processed data will be moved safely to the archive in compressed format.

4. Data Structures

4.1 PLC data format

4.2 OPC standards

4.3 Parquet data format

4.4 Influx DB Structure

The InfluxDB will be used as our primary DBMS.

Some basic terminologies:

- Schema - How the data are organized in InfluxDB. The fundamentals of the InfluxDB schema are databases, retention policies, series, measurements, tag keys, tag values, and field keys. See Schema Design for more information.
- Retention Policy (RP) - Describes how long InfluxDB keeps data (duration), how many copies of the data to store in the cluster (replication factor), and the time range covered by shard groups (shard group duration). RPs are unique per database and along with the measurement and tag set define a series. When you create a database, InfluxDB creates a retention policy called autogen with an infinite duration, a replication factor set to one, and a shard group duration set to seven days.
- Timestamp - The date and time associated with a point. All time in InfluxDB is UTC but can be changed.
- Measurement - The part of the InfluxDB data structure that describes the data stored in the associated fields. Measurements are strings.
- Tag - The key-value pair in the InfluxDB data structure that records metadata. Tags are an optional part of the data structure, but they are useful for storing commonly-queried metadata; tags are indexed so queries on tags are performant. Query tip: Compare tags to fields; fields are not indexed.
- Tag key - The key part of the key-value pair that makes up a tag. Tag keys are strings and they store metadata. Tag keys are indexed so queries on tag keys are performant.
- Tag set - The collection of tag keys and tag values on a point.
- Tag value - The value part of the key-value pair that makes up a tag. Tag values are strings and they store metadata. Tag values are indexed so queries on tag values are performant.
- Field - The key-value pair in an InfluxDB data structure that records metadata and the actual data value. Fields are required in InfluxDB data structures and they are not indexed - queries on field values scan all points that match the specified time range and, as a result, are not performant relative to tags.
- Field key - The key part of the key-value pair that makes up a field. Field keys are strings and they store metadata.
- Field set - The collection of field keys and field values on a point.
- Field value -The value part of the key-value pair that makes up a field. Field values are the actual data; they can be strings, floats, integers, or Booleans. A field value is always associated with a timestamp. Field values are not indexed - queries on field values scan all points that match the specified time range and, as a result, are not performant.

InfluxDB Schema for /data/* and /archive/* feed records

For /data we will be having a bucket names cam_feeds with a retention policy of about 1 week. Under cam_feeds measurements of raw_feeds and calc_feeds will be there. We will be using feed_id as a tag key so that it can be indexable like a primary key. The field key will be used as key=value pair to store our records like feed_title, start_time, end_time, vid_path, and from_cam. After 1 week i.e., our retention policy, all data will be transferred from raw_feeds > arch_raw_feeds and calc_feeds > arch_calc_feeds.

Buckets	cam_feeds (Retention policy = 1 week)
Measurements	raw_feeds, calc_feeds, arch_raw_feeds, arch_calc_feeds
Tags Keys	feed_id (String, index)
Field Keys	• feed_title (String) • start_time (unixtime) • end_time (unixtime) • vid_path (String) • from_cam (String)

Table 1 cam_feeds

Let us take a sample data for only 1 raw feed for /data:

_time	**_measurement**	**feed_id**	**_field**	**_value**
2022-01-01T00:00:00Z	raw_feeds	3eqenu3e3	feed_title	brm_stand1_therm_cam_1641062400_1641062700.mp4
2022-01-01T00:00:00Z	raw_feeds	3eqenu3e3	start_time	1641062400
2022-01-01T00:00:00Z	raw_feeds	3eqenu3e3	end_time	1641062700
2022-01-01T00:00:00Z	raw_feeds	3eqenu3e3	vid_path	/data/raw/202209/640975400/
2022-01-01T00:00:00Z	raw_feeds	3eqenu3e3	from_cam	stand_1_thermal_cam

Table 2 sample data for /data

Schema:

```
raw_feeds, feed_id=3eqenu3e3,
feed_title=brm_stand1_therm_cam_1641062400_1641062700.mp4, start_time=1641062400,
end_time=1641062700, vid_path=/data/raw/202209/640975400/,
from_cam=stand_1_thermal_cam
```

Same for calc_feeds:

```
calc_feeds, feed_id=3eqenu3e3,
feed_title=brm_stand1_therm_cam_1641062400_1641062700.mp4, start_time=1641062400,
end_time=1641062700, vid_path=/data/calc/202209/640975400/,
from_cam=stand_1_thermal_cam
```

InfluxDB Schema for sensor data

Here we will be having all the data which are available from the installed sensors. Along with the other fields like _time, sensor_name, sensor_zone, sensor_type, and sensor_value.

Buckets	snsrs_data (Retention policy = 1 week)
Measurements	data
Tags Keys	snsr_id (String, index)
Field Keys	• snsr_name (String) • snsr_zone (String) • snsr_type (String) • snsr_value (String)

Table 3 snsrs_data

Let us take 1 sample data for sensor_data:

_time	**_measurement**	**snsr_id**	**_field**	**_value**
2022-01-01T00:00:00Z	data	89d5a82s	snsr_name	infrared_sensor_xyz_inc_bsp
2022-01-01T00:00:00Z	data	89d5a82s	snsr_zone	brm_stand_16
2022-01-01T00:00:00Z	data	89d5a82s	snsr_type	infrared_snsr
2022-01-01T00:00:00Z	data	89d5a82s	snsr_value	846.694546221949562

Table 4 sensor_data

Schema:

```
data, snsr_id=89d5a82s, snsr_name=infrared_sensor_xyz_inc_bsp, snsr_zone=brm_stand_16,
snsr_type=infrared_snsr, snsr_value=846.694546221949562
```

As influxDB will automatically input a _time whenever the record is entered, we will not have a different field for a timestamp and I'm assuming that there is no delay between the actual timestamp when a sensor has taken a value and the value of timestamp when the recorded is inserted, although ill make some tests.

InfluxDB Schema design for abnormal pattern records

We are going to store records which are abnormal or fall under our defined abnormal pattern category. This data will help us in future regarding the analysis and study of the data wherever, when and how the cobble or any other incident has happened.

Buckets	abnormal_pattern (Retention policy = 1 week)
Measurements	data
Tags Keys	patt_id (String, index)
Field Keys	• patt_type (String) • patt_snsr (String) • patt_zone (String) • patt_occ_at (unixtime) • patt_occ_till (unixtime)

Table 5 abnormal_pattern

Let us take 1 sample data for abnormal_pattern:

_time	**_measurement**	**pattern _id**	**_field**	**_value**
2022-01-01T00:00:00Z	data	5sd5sd68	patt_type	cobble_med_brm
2022-01-01T00:00:00Z	data	5sd5sd68	patt_snsr	infrared_snsr
2022-01-01T00:00:00Z	data	5sd5sd68	patt_zone	brm_stand_16
2022-01-01T00:00:00Z	data	5sd5sd68	patt_occ_at	1641062400
2022-01-01T00:00:00Z	data	5sd5sd68	patt_occ_till	1641062700

Table 6 Sample data for abnormal_pattern

Schema:

```
data, patt_id=89d5a82s, patt_type=cobble_med_brm, patt_zone=brm_stand_16,
patt_occ_at=1641062400, patt_occ_till=1641062400
```

Workaround to relate records with InfluxDB

As I have mentioned that influxDB is not an RDBMS and because of that relations aren't possible with influxDB. I was thinking to make a separate index for our available camera, sensors, and defined patterns (events) in our schema based on JSON. This will help us to have an index of those along with their IDs which are also tag keys in our influxDB records.

JSON schema sample for the camera:

```
{
    "cam_id": "5asd55as5",
    "cam_name": "thermal_cam_sony_inc",
    "cam_type": "thermal",
    "cam_zone": "brm_stand_16",
    "usr_note": "This camera is for xyz purpose",
    "specs": {
        "sensor_technology": "CMOS",
        "supported_interface": "GigE Vision",
        "spectrum_capability": "Visible (400-700 nm), Near Infrared (700-1000 nm)",
        "spectrum": "MONO/NIR",
        "pixel_size": "4.8 µm",
        "resolution": "5120 x 5120",
        "max_frame_rate": "51fps",
        "vendor": "Sony Visual Imaging Solutions"
    }
}
```

Figure 7 JSON schema sample for the camera

JSON schema sample for our available sensors:

```
{
      "snsr_id": "9b9t2t5f",
      "snsr_name": "gas_snsr_ada_inc",
      "snsr_type": "Gas Sensor",
      "snsr_zone": "brm_stand_16",
      "snsr_note": "This sensor is for xyz purpose",
      "specs":
            {
            "communication_interface": "I2C/PLC/OPC",
            "measuring_range": "1 lux to 65535 lux",
            "precision": "±(50+5% reading)",
            "operating_voltage": "4.5 ~ 20",
            "operating_condition": "-10℃~+50℃; 0~95%RH (non-condensing)",
            "delay_time": "Default 8S + -30% (can be customized)",
            "vendor": "Adafruit"
      }
}
```

Figure 8 JSON schema sample for the camera

JSON schema sample for our pre-defined events:

```
{
      "patt_id": "9b9t2t5f",
      "patt_name": "gas_snsr_ada_inc",
      "patt_type": "Gas Sensor",
      "patt_note": "This sensor is for xyz purpose",
      "patterns":
            {
                  "by_senors":
            [
                {
                    "snsr_id": "56sd65363",
                    "snsr_def_th": "55.45455956",
                    "snsr_rec_th": "89.56556565"
                },
                {
                    "snsr_id": "9sd8ee5w2",
                    "snsr__defth": "95.14245544",
                    "snsr_rec_th": "89.56556565"
                }
            ]
            },
    "patt_time_at": "1641062400",
    "patt_time_till": "1641062700"
}
```

Figure 9 JSON schema sample for our pre-defined events

Note: If we need to get more data about the sensor, camera, and more about events we can get it from these indexes using the IDs which are used as tags in our influxDB.

Security: In influxDB, we have token/password-based authentication for security, but to keep these index data secure we can limit permissions for the user using the UNIX system permission concept also we can encrypt these data using Encryption methods like AES. Encryption, Integrity, Security etc. are very important to us and I'll write and explain more about them later in the other document.

5. PLC Analytical System

5.1 Obtaining data from PLC signals

5.2 Processing of Data

For conversion of CSV/TSV to Apache Parquet

As we have decided to use apache's parquet instead of CSV/TSV which is slow and uses more storage compared to apache parquet which is an open-source, column-oriented data file format designed for efficient data storage and retrieval. This tool will allow us to convert our existing or old CSV/TSV data files to apache's parquet with various other configurable parameters in a config file.

Working: It uses the native implementation of apache's arrow and parquet in rust to read, decode and convert it. Using a config file which will be read and decoded by a shell script. It reads the users defined configuration from the *config.config* file and parse it as parameters to the converter.

Config.config structure: There can be multiple config files for multiple sensors or as per requirements, but the program will try to read from .configs/config.config.

Sample config file:

```
# Config file for the CSV/TSV to parquet conversion tool.
# The name of the config file.
NAME="bsp_zone1_snsr_1-9"
# The description of the config file.
DESCRIPTION="This config is used for xyz purpose at BSP."
# Path to the csv file
PATH_TO_CSV="abc.csv"
# Path where the parquet file should be saved
PATH_TO_PQT="abc.parquet"
# Set the CSV file's column delimiter as a byte character [default: ,]
#DELIMITER="'\t'"
# created_by tag for the file
CREATED_BY="BSP_BRM"
# The number of records to infer the schema from.
#MAX_READ_RECORDS="100"
# If the CSV file contains header
#HEADER=true
# Sets flag to enable/disable dictionary encoding for any column
```

```
DICTIONARY=true

# Compression

# Set the compression [possible values: uncompressed, snappy, gzip, lzo, brotli, lz4, zstd]

#COMPRESSION="uncompressed"

# Sets encoding for any column.

# [possible values: plain, rle, bit-packed, delta-binary-packed, delta-length-byte-array, delta-byte-array, rle-dictionary]

#ENCODING="plain"

# Sets data page size limit

#DATA_PAGE_SIZE_LIMIT=""

# Sets dictionary page size limit

#DICTIONARY_PAGE_SIZE_LIMIT=""

# Sets write batch size

#WRITE_BATCH_SIZE=""

# Sets max size for a row group

#MAX_ROW_GROUP_SIZE=""

# Sets flag to enable/disable statistics for any column [possible values: none, chunk, page]

#STATISTICS="none"

# Sets max statistics size for any column. Applicable only if statistics are enabled

#MAX_STATISTICS_SIZE=""

# Print the schema as output in the terminal

SCHEMA=true
```

Fields

- Name: The name of the config file. As there can be multiple config files, naming it will help us to differentiate.
    - String: A string value.
- Description: Along with the name there can be a small description for the file where the purpose and other things can be described.
    - String: A string value.
- Path_to_csv: The path where the CSV/TSV file is located, should be inside ' '.
    - String: A string value.
- Path_to_pqt: The path where you want to save the parquet file, should be inside ' '. Also, the name of the file along with the extension (.parquet) should be in the path.

    - String: A string value

- created_by: Add created_by tag for the parquet file.
    - String: A string value.

- max_read_records: The maximum number of records to be read from the CSV/TSV file.
    - usize: should be an Unsigned Integer of value, usize types depend on the architecture of the computer your program is running on, which is denoted in the table as “arch”: 64 bits if you’re on a 64-bit architecture and 32 bits if you’re on a 32-bit architecture.

- header: Whether the CSV file contains the header or not.
    - Bool: Either True or False.

- dictionary: Sets flag to enable/disable dictionary encoding for any column.
    - Bool: Either True or False.

Note: The raw signals data which was exported directly from the ibaAnalyser as CSV needs to have same-length records else it will throw an uneven record error and the file will not be converted as parquet.

Let’s say there is a sample CSV file:

```
col1, col2, col3, col4, col5

val1, val2, val3, val4

val5, val6, val7, val8
```

Here we can see that our CSV have 5 columns but the values for col5 are NULL and can cause an error while converting. We have noticed that this was happening in all of the CSV files that we exported from the ibaAnalyser. To overcome from this we need to make sure the CSV files have same-length records and for that, we have a tool (xsv) which is installed on our server which can force the file to have the same-length record either by padding or truncating them.

Documentation for csvtool: for indexing, slicing, analysing, splitting and joining CSV

This program will be used by us to analyse the CSV/TSV files like mean, avg, median, sum, max/min, cardinality and other stats. Also will allow us to modify the CSV/TSV files like re-ordering the columns, and rows, slicing, joining, splitting etc.

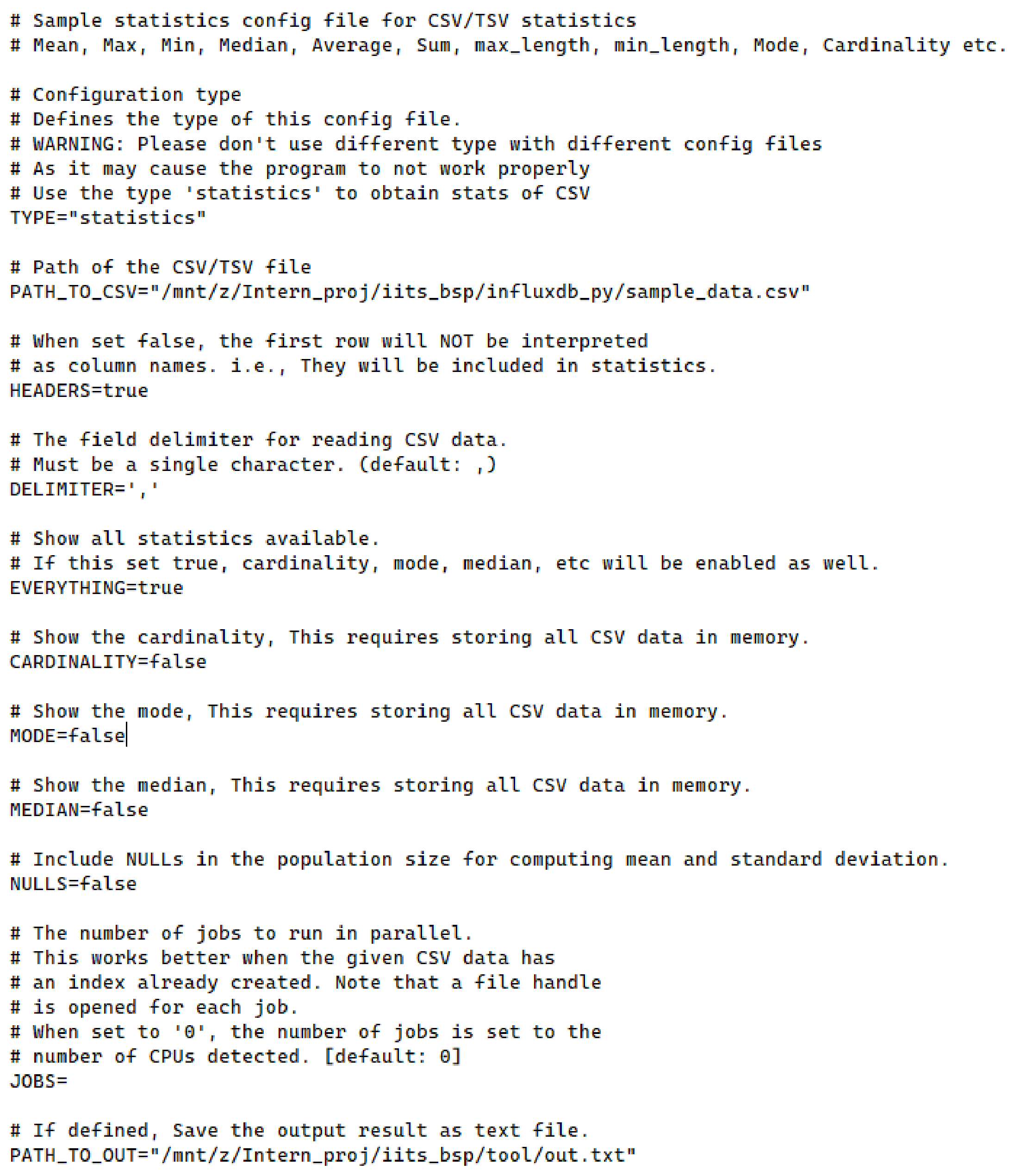
As we have noticed that we have a large number of records in our CSV files, so we need a fast and efficient tool to read, organise and perform other operations in our CSV file for that, we will be using a tool name xsv.

xsv will allow us to create an index before performing any operations on the file so that we can perform slicing, splitting, and gathering statistics much faster. It will create an index file on the same path as input.csv.idx The index will be automatically used by commands that can benefit from it. If the original CSV data changes after the index is made, commands that try to use it will result in an error (you have to regenerate the index before it can be used again).

To create an index for the CSV file:

```
xsv index path_to_csv
```

xsv can also generate some statistics like sum, min, max, mean, stddev, median, mode, cardinality etc



```
# Sample statistics config file for CSV/TSV statistics
# Mean, Max, Min, Median, Average, Sum, max_length, min_length, Mode, Cardinality etc.

# Configuration type
# Defines the type of this config file.
# WARNING: Please don't use different type with different config files
# As it may cause the program to not work properly
# Use the type 'statistics' to obtain stats of CSV
TYPE="statistics"

# Path of the CSV/TSV file
PATH_TO_CSV="/mnt/z/Intern_proj/iits_bsp/influxdb_py/sample_data.csv"

# When set false, the first row will NOT be interpreted
# as column names. i.e., They will be included in statistics.
HEADERS=true

# The field delimiter for reading CSV data.
# Must be a single character. (default: ,)
DELIMITER=','

# Show all statistics available.
# If this set true, cardinality, mode, median, etc will be enabled as well.
EVERYTHING=true

# Show the cardinality, This requires storing all CSV data in memory.
CARDINALITY=false

# Show the mode, This requires storing all CSV data in memory.
MODE=false

# Show the median, This requires storing all CSV data in memory.
MEDIAN=false

# Include NULLs in the population size for computing mean and standard deviation.
NULLS=false

# The number of jobs to run in parallel.
# This works better when the given CSV data has
# an index already created. Note that a file handle
# is opened for each job.
# When set to '0', the number of jobs is set to the
# number of CPUs detected. [default: 0]
JOBS=

# If defined, Save the output result as text file.
PATH_TO_OUT="/mnt/z/Intern_proj/iits_bsp/tool/out.txt"
```

Figure 10 Sample statistics config file

Fields

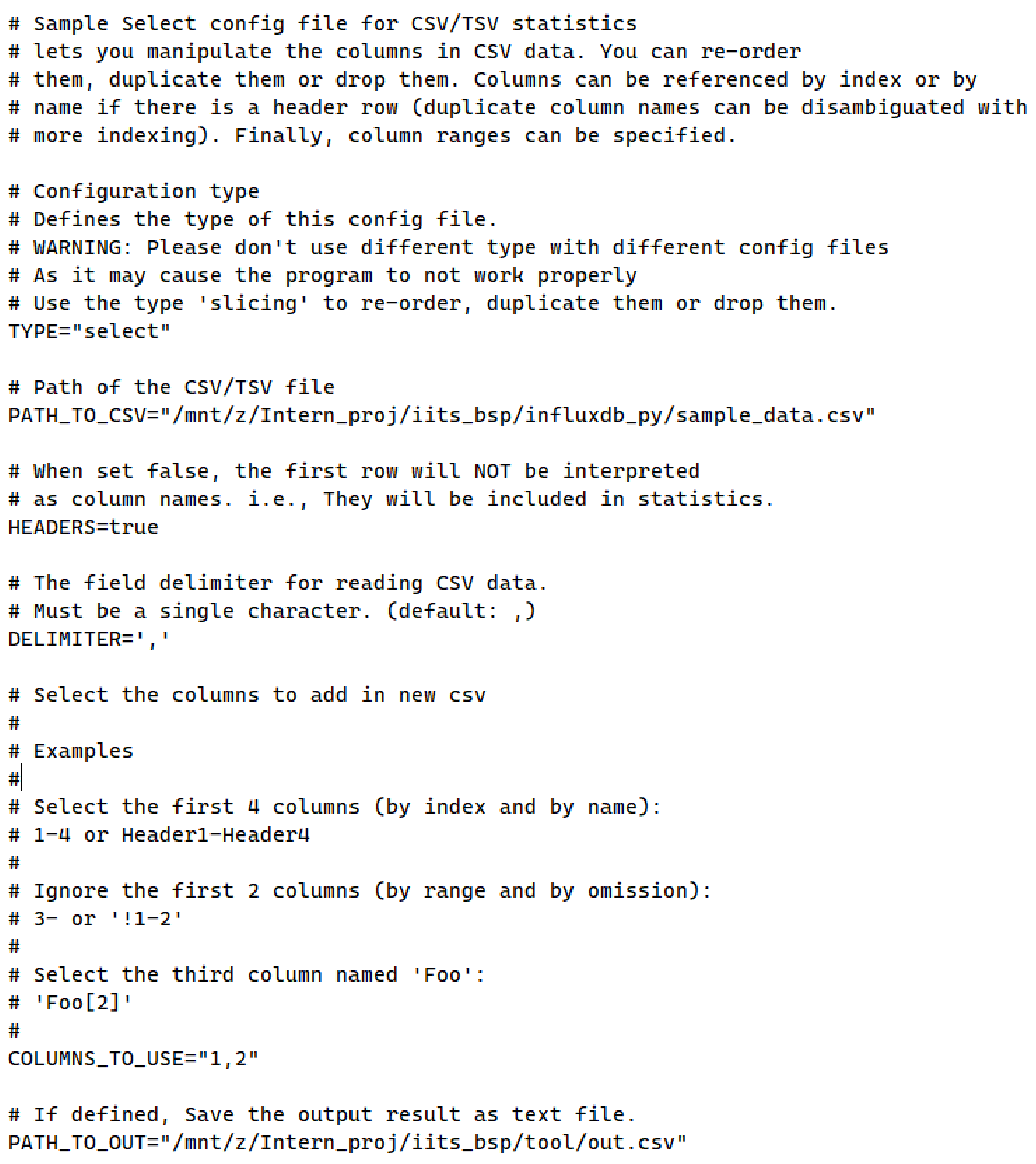
- TYPE: A String which denotes the work which needs to be done by the program, please don’t use different types with different defined work.

- PATH_TO_CSV: A path to the CSV/TSV file which will be used to generate stats.

- HEADERS: Takes a Boolean value, if set to false then the first row will be interpreted as statistics, not as columns.

- DELIMITER: A char value which is the field delimiter for the CSV/TSV data.

- EVERYTHING: Show all statistics available. If this is set true, cardinality, mode, median, etc. will be enabled as well.
- CARDINALITY: Show the cardinality, this requires storing all CSV data in memory.
- MODE: Show the mode, this requires storing all CSV data in memory.
- MEDIAN: Show the median, this requires storing all CSV data in memory.
- NULLS: Include NULLs in the population size for computing mean and standard deviation.
- JOBS: The number of jobs to run in parallel. This works better when the given CSV data has an index already created. Note that a file handle is opened for each job. When set to '0', the number of jobs is set to the number of CPUs detected. [Default: 0]
- PATH_TO_OUT: If defined, Save the output result in a text file.



```
# Sample Select config file for CSV/TSV statistics
# lets you manipulate the columns in CSV data. You can re-order
# them, duplicate them or drop them. Columns can be referenced by index or by
# name if there is a header row (duplicate column names can be disambiguated with
# more indexing). Finally, column ranges can be specified.

# Configuration type
# Defines the type of this config file.
# WARNING: Please don't use different type with different config files
# As it may cause the program to not work properly
# Use the type 'slicing' to re-order, duplicate them or drop them.
TYPE="select"

# Path of the CSV/TSV file
PATH_TO_CSV="/mnt/z/Intern_proj/iits_bsp/influxdb_py/sample_data.csv"

# When set false, the first row will NOT be interpreted
# as column names. i.e., They will be included in statistics.
HEADERS=true

# The field delimiter for reading CSV data.
# Must be a single character. (default: ,)
DELIMITER=','

# Select the columns to add in new csv
#
# Examples
#
# Select the first 4 columns (by index and by name):
# 1-4 or Header1-Header4
#
# Ignore the first 2 columns (by range and by omission):
# 3- or '!1-2'
#
# Select the third column named 'Foo':
# 'Foo[2]'
#
COLUMNS_TO_USE="1,2"

# If defined, Save the output result as text file.
PATH_TO_OUT="/mnt/z/Intern_proj/iits_bsp/tool/out.csv"
```

Figure 11 Sample select config file

Fields

- TYPE: A String which denotes the work which needs to be done by the program, please don't use different types with different defined work.

- PATH_TO_CSV: A path to the CSV/TSV file which will be used to generate stats.

- HEADERS: Takes a Boolean value, if set false then the first row will be interpreted as statistics, not as columns.

- DELIMITER: A char value which is the field delimiter for the CSV/TSV data.

- COLUMNS_TO_USE: Selects the columns of the file which should be used.
    - Select the first 4 columns (by index and by name): 1-4 or Header1-Header4
    - Ignore the first 2 columns (by the range and by omission): 3- or '!1-2'
    - Select the third column named 'Foo': 'Foo[2]'
- PATH_TO_OUT: If defined, Save the output result as a new CSV file.

5.3 Data Filtering and analysis

5.4 Data conversion and archiving

5.5 Data to time series database

6. Video Analytical System

6.1 GigE vision standards and design

6.2 GigE feeds storage and analysis logic

Acronyms

PLC	**Programmable Logic Controller**
OPC	**Open Protocol Communication**
CSV	**Comma Separated Values**
UTF8	**UCS (Unicode) Transformation Format**
ZSTD	**Z Standard**
RP	**Retention Policy**
ZFS	**Zettabyte File System**
RAID	**Redundant Array of Inexpensive Disks**
TSDB	**Time Series Data Base**