Monitoring the readout temperature in the ICARUS TPC

2023 Italian Students Program
Final Report

Giovanni Zago*
Physics of Data student, Università di Padova, Padova, Italy

Filippo Varanini
INFN Sezione di Padova and Università di Padova, Padova, Italy

Geoff Savage
Fermi National Accelerator Laboratory, Batavia, IL 60510, USA

(Dated: August - September 2023)

Abstract

Being a Liquid Argon Time Projection Chamber (LAr-TPC) detector, ICARUS exploits a series of three parallel wire planes to measure and collect the ionization charge coming from an event inside the active volume. The wires are read out by A2795 custom boards, arranged in crates located on top of the two liquid Argon vessels. Each board is provided with two temperature sensors that allow temperature monitoring while the experiment is running: this is key in order to ensure data reliability and maintain hardware in safe conditions. Moreover, temperature-related problems may force unplanned shutdowns that result in losing valuable time for the experiment to run. In this work we are discussing the characterization of the ICARUS TPC board temperatures and the system we implemented in order to monitor them effectively. Among the tools used, Grafana results in a valid application for both displaying temperature data and sending alerts to the shifters whenever a pathological situation occurs.

* giovanni.zago.2@studenti.unipd.it

I. INTRODUCTION

ICAURS [1] is one of the two experiments that constitute the Short Baseline Neutrino (SBN) Program at Fermilab. The goal of the SBN program is to study the neutrino physics with a particular focus on neutrino oscillations and Beyond Standard Model searches, like investigating the existence of the sterile neutrino. In the near future, ICARUS is meant to be operating together with the Short Baseline Neutrino Near Detector (SBND) experiment by sharing the same neutrino beam (Booster Neutrino Beam).

ICARUS is a Liquid Argon Time Projection Chamber (LAr-TPC) consisting in two identical adjacent modules (Fig. 1, (1)), having a central cathode and two anodic planes, each one made up by 3 layers of 100 μm-diameter, 9 m-long wires separated 3 mm from each other (Fig. 1, (2)). The first two wire planes provide a nondestructive charge measurement, while the the third one collects all the ionization charge. There is a total of 53,248 wires inside the detector that are read out by custom A2795 boards [2], arranged inside crates (Fig. 1, (3)) hosted on flanges on top of each vessel. Crates are numbered from EE01 to WW20 as shown in Fig. 1 (4). Flanges numbered 01 or 20 actually consist in 3 stacked crates (Fig. 2, left) so, in this case, in order to specify a single crate, it is necessary to add a trailing T (top), M (middle), B (bottom) (e.g. to indicate the top crate of the EE01 flange we use EE01T). A2795 boards embed both the analogue front end and a digital module implemented on a powerful FPGA. Moreover, the boards have two temperature sensors: one is located near the preamplifier modules and the other near the regulator, which is the most power-consuming component inside the board. The information about the temperature of the boards is key when dealing the gain/noise ratio of the signal coming from the wires. In particular, the charge preamplifiers are the most sensitive components to temperature variations. In light of these considerations, in this work we are showcasing the results of the brief study we have conducted for implementing an efficient system for both characterizing and preserving the board temperature data. Thus, our goals can be summarized as follows:

- Study the data flow needed for retrieving the desired data. Then, process the data in order to make it easily usable and accessible.
- Characterize the distribution of the board temperatures in normal conditions by looking at data coming from the past experiment runs. Then, study selected runs showing pathological behaviour of the board temperatures to identify under which conditions it is reasonable to fire an alarm to the shifters.
- Use the knowledge gained from the data analysis to implement the alarms on a online monitoring platform.

In order to fulfill the first goal we had to focus on studying the data flow of the temperature, i.e. the path needed to get each piece of data from the source - the board temperature sensors - to the final user, who can eventually perform high-level manipulation (i.e. analysis, plots, etc.). This part of the work will be discussed in Section II. The second goal consists in using the polished temperature data to perform an analysis of both normal and pathological runs. This analysis will be addressed in Section III. The third goal consists in identifying key features from the data analysis that can be easily used as alarming thresholds/conditions to embed within a monitoring application like Grafana [3]. This will be addressed in Section IV. In Section V we provide suggestions about tools one has to use to easily reproduce the analysis we have performed in one of the previous sections, or to perform further analysis

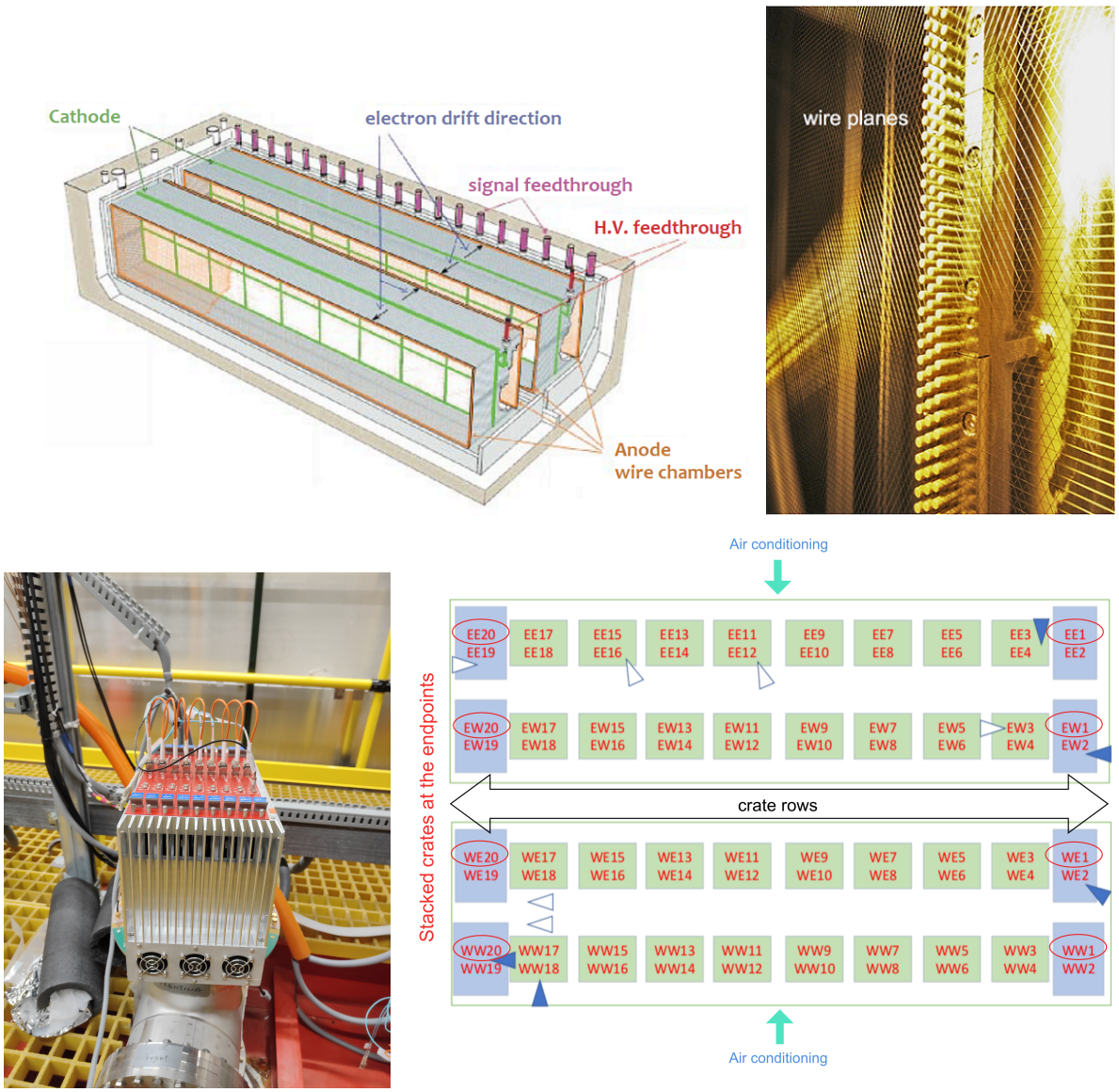

FIG. 1. *From top-left to bottom-right*: (1): 3D model of ICARUS, in which it is possible to see that the experiment consists of two vessels that share the same structure, with the cathode in the middle and two anodic wire planes that extend along the main side of the vessel. (Image taken from [4]) (2): a picture showing a detail of the wire planes. The horizontal wires are the Induction 1 plane, while the Induction 2 and Collection planes wires are arranged with an angle of $+60°$ and $-60°$ with respect to the horizontal direction respectively. (Image taken from [4]) (3): a picture showing a crate, mounted onto a flange, hosting 9 A2795 boards. (4): schema of the crate arrangement on top of the two vessels. Each green or blue box corresponds to two crates like the one shown in the previous figure, with the exception of those numbered 1 or 20 that are stacked crates (cf. Fig. 2, left).

with the goal of expanding or improving the work that has already been done. Finally, Section VI is left to the conclusions.

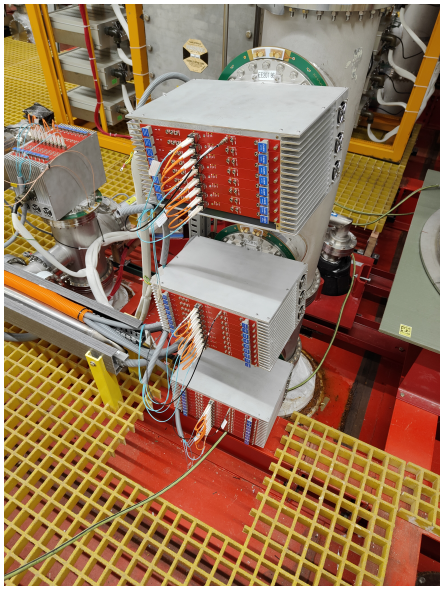

FIG. 2. *Left*: flange with 3 stacked crates. *Righ*: corridor that runs along the main side of the West module. It is possible to spot on the right the white air conditioning outlets that throw fresh air directly on the middle of the crate row.

II. RETRIEVING AND PROCESSING THE DATA

A. Starting point of the board temperature data flow

The board temperature sensors are read out by a DAQ software (artdaq [5]) process called `BoardReader`, which sends the data to a Graphite [6] back-end, consisting in so-called Carbon daemons that push data in a database. However, this database is in the *Whisper* format, thus it is fix-sized and consists in a series of *archives* characterized by a specific resolution and data retention settings. This means that the data stored inside the Graphite database is subject to some kind of aggregation that happens over time in order to reduce its granularity and keep the database size unchanged. This also implies that the data is deleted from the database once a certain amount of time has elapsed since the data collection. It is clear that this kind of database is a good tool for monitoring purposes but it is not the optimal choice when data analysis is required. Indeed, data compression and aggregation make it impossible to spot the rapid (approximately, over a few minutes) temperature variations we are interested on. Moreover, the APIs provided by Graphite do not allow an easy and effortless access to the database in order to perform high-level computations with the most common tools (Pandas, NumPy, etc.). A possible way to retrieve clean and uncompressed temperature data from the boards may be to modify the artdaq code, so as to send the metrics to another data consumer that does not perform the manipulation provided by Graphite. However, this solution would require a great effort, since it would be necessary to setup again the boards in order for them to adapt to the software modifications. A more immediate solution - which we followed - consists in retrieving the `BoardReader` log files which contain all the temperature values and the metadata needed to identify an unique measurement. The `BoardReader` log files are shared on the ICARUS Network File System (NFS) and their paths, for each run, are specified inside the `metadata.txt` file

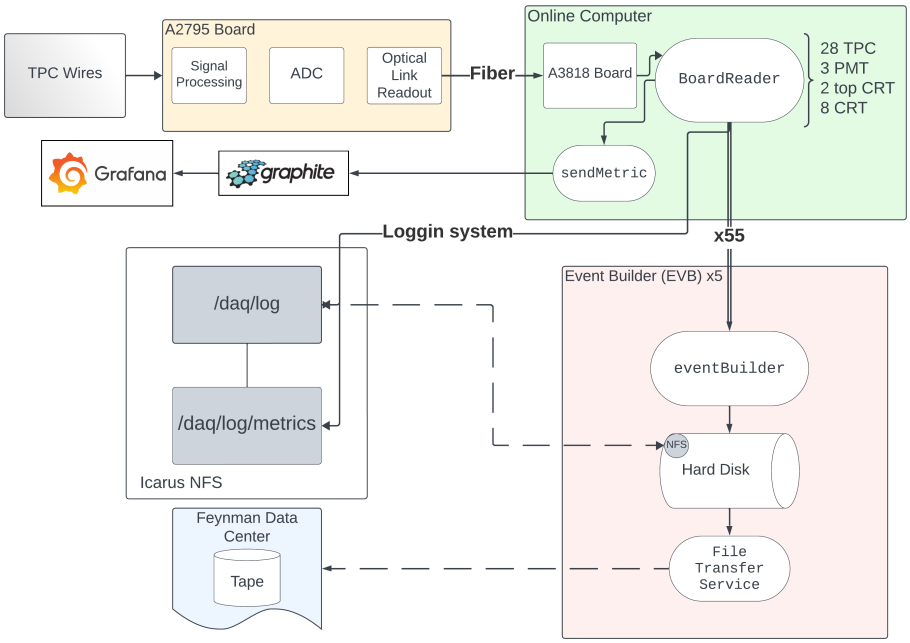

FIG. 3. Schema of the connections among hardware components and related software processes inside ICARUS.

located inside the run dedicated directory, which is `/daq/run_records/RUN_ID/`. It turns out that all the `BoardReader` log files are stored inside sub-directories of `/daq/log/`, like `/daq/log/icarustpcee20m-icarus-tpc19-11170/`. There are 96 `BoardReader` log files - one for each board - and each of them contains the path of the corresponding metric log file where the actual temperature values are reported: all these metric log files are inside the `/daq/log/metrics/` directory (cf. Fig. 3).

B. Data processing scripts

In order to process the raw metric log files we have developed a series of scripts that allow not only to perform a high-level analysis relying on polished data, but also store the temperature data inside the ICARUS Postgres online production database (`icarus_online_prd`) to ensure a long term, granular preservation. The scripts are embedded inside a cronjob that runs continuously on one of the ICARUS online machines, thus ensuring that the board temperature data is automatically pushed inside the database without needing an active supervision. All the scripts are currently stored inside the NFS directory `/home/nfs/icarus/gzago/` and are organized in the following way:

- The script that runs as a cronjob is `cronjob.sh`. It launches the `tpc_readout_temperatures.sh` script and also creates a cronjob log file inside the `/home/nfs/icarus/gzago/cjob_logs/` directory.
- `tpc_readout_temperatures.sh` is the main script that embeds all the other scripts. While running the following scripts are run in sequence:

- `runs_stored.py` connects to the database and queries for the `RUN_ID` of the last run stored inside it. This is important because the cronjob is supposed to start pushing the board temperature data inside the database only when the considered run has finished. A coarse but effective way to check that is to see if the following run has started, i.e. checking for the existence of the directory `/daq/run_records/RUN_ID+1/`.
- `find_logs_v3.sh` fetches all the metric log files paths for that run and prints them inside a text file inside `/home/nfs/icarus/gzago/` with the name `metric_paths_run_RUN_ID_v3.txt`.
- `logfile_parser_script_v11.py` takes every metric log file, parses it and pushes the data into the database.
- `tpc_readout_temperatures_update_runs.py` marks the processed run as "stored" inside the database. At this point, the cronjob is completed.

A comprehensive schematic outlook of the data flow is reported on Fig. 4.

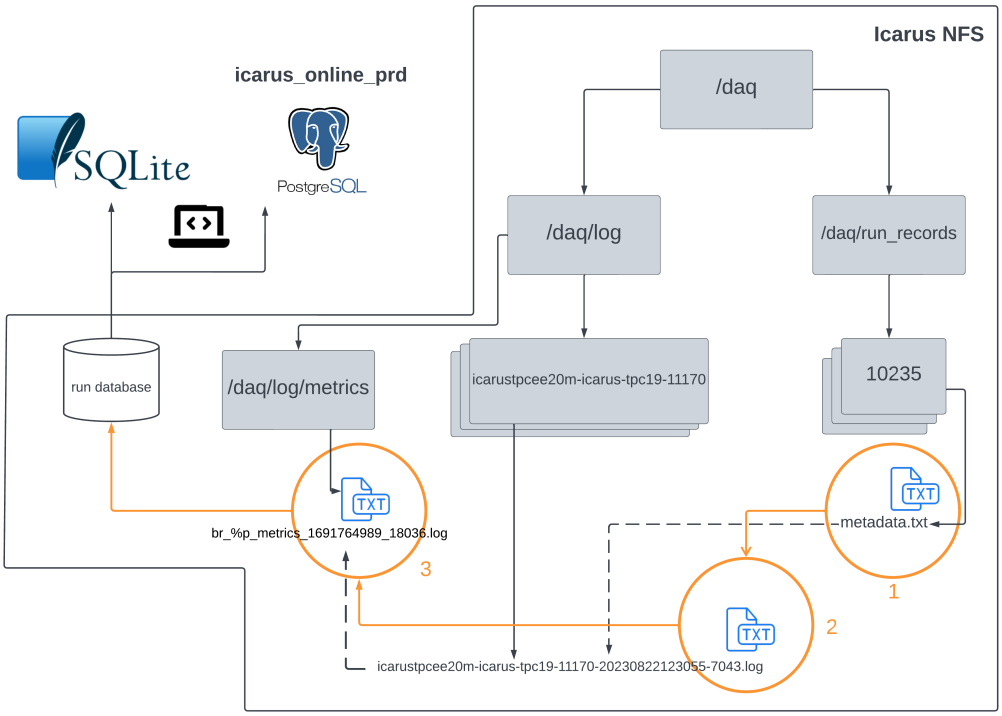

FIG. 4. Schema of the directories inside the ICARUS NFS. The orange circles and arrows represent the data flow that starts from collecting the `metadata.txt` file (step 1) for the selected run, continues with collecting the `BoardReader` log files and the metric log files (steps 2 and 3) and ends with storing the data inside the `icarus_online_prd` database (or a SQLite database, if desired). Black solid lines represent the hierarchical location of directories and files, while black dashed lines represent references (paths) from a file to another one.

C. Table structure and management

The table inside the ICARUS online production database to which the data is fed is `dcs_prd.tpc_readout_temperatures` and has the following schema:

Column	Type
index	integer
run	integer
location	character varying(10)
number	integer
stack	integer
timestamp	timestamp with time zone
board	integer
temp_id	integer
value	real

TABLE I. `dcs_prd.tpc_readout_temperatures` table schema.

Of course, all these columns are needed to locate each temperature measurement both in space (cf. Fig. 1, bottom-right) and time. In the following we describe in detail the columns of the table:

- `index`: index of the row of the table.
- `run`: it corresponds to the `RUN_ID` number that univocally identifies a run.
- `location`: this field specifies which of the four rows the crate that we are considering belongs to. In accordance with the convention reported in Sec. I, valid values for `location` are `ee`, `ew`, `we`, `ww`.
- `number`: this field specifies the number of the crate inside the row. Valid values are integers between 1 and 20.
- `stack`: it specifies if the considered crate is mounted alone onto the flange (crates with `number` from 2 to 19) or belongs to a three-crate flange (crates with `number` 1 or 20). Since `stack` is an integer value, the matching is the following: top crate $\rightarrow$ `1`, middle crate $\rightarrow$ `2`, bottom crate $\rightarrow$ `3`.
- `timestamp`: it is the temporal coordinate of the measurement. The format is `YYYY-mm-dd HH:MM:SS-05:00`, where the subtracted hours at the end specify the time zone with respect to the GMT reference.
- `board`: number of the board inside the crate.
- `temp_id`: it is 1 or 2 according to which temperature sensor the measurement comes from.
- `value`: the actual temperature measurement.

The table has also to keep track of the runs that are stored inside the database: this is implemented by the `tpc_readout_temperatures_update_runs.py` script (cf. Subsec. II B) that adds a line to the table with the structure reported on Tab. II.

index	run	location	number	stack	timestamp	board	temp_id	value
AUTOINCREMENT	RUN_ID	stored	0	0	CURRENT_TIMESTAMP	0	0	0

TABLE II. Structure of the rows that mark that the data of a run identified by RUN_ID has been processed and pushed inside the database.

D. Error handling

During the development of the scripts we have considered implementing a basic error-handling system. Since the data flow involves different elements inside the ICARUS online network, there are multiple gears that can fail while executing the whole script chain. Moreover, there can be, of course, issues not related to the execution of the code, like a sudden power outage or a crash of the machines on which the scripts are supposed to run. The error-handling system is managed by `tpc_readout_temperatures.sh` through checks of the exit codes of the other scripts: each scripts prints inside the cronjob log file its own error message, which is then reiterated by a message of `tpc_readout_temperatures.sh` itself as a double check. This redundancy has the goal of facilitating the debugging of the code by having scripts that can be run and that can provide understandable outputs independently from each other. Regarding the errors, reported in Tab. III, we remark that if there is an interruption of the script that pushes the data inside the database - i.e. `logfile_parser_script_v11.py` -, then, at the following cronjob run, the slice of data already pushed inside the database will be deleted, in order to start with a clean data push the second following run after the one in which the crash happened.

Script	Error message	Source	Comment
`runs_stored.py`	Job interrupted because of a failure in retrieving the last run stored: check database connection	S/M	S returns exit code 1. M terminates the script.
	Job interrupted because of a generic error in retrieving the last run stored.	M	M returns any exit code that is not 0. M terminates the script.
`find_logs_v3.py`	Metadata text file does not exists.	S/M	S returns exit code 1. M marks the run as completed and terminates the script.
	Job interrupted because the run directory redirects to no tpc boardreader log files or the boardreader log files are unavailable.	S/M	S returns exit code 2. M marks the run as completed and terminates the script.
	Job interrupted because no metric log file has been found for the current run.	S/M	S returns exit code 3. M marks the run as completed and terminates the script.
	Job interrupted because of a generic error occurred while running `find_logs_v3.sh`.	M	S returns any exit code that is not 0. M marks the run as completed and terminates the script.

`logfile_parser_script_v11.py`	Job interrupted owing to database connection fail.	S/M	S returns exit code 1. M terminates the script.
	Job interrupted because run `RUN_ID` data was already present on the db.	S/M	S returns exit code 2. M terminates the script. Data of run `RUN_ID` gets deleted from the database.
	Job interrupted because run `RUN_ID` existence verification failed	S/M	S returns exit code 3. M terminates the script.
	Job interrupted because of a generic error occurred while running `logfile_parser_script_v11.py`	S/M	M returns any exit code that is not 0. M terminates the script.

TABLE III: Scripts error messages. Inside the Source column, S/M means that the error message comes first from the script and then is reiterated by the main script M. The indication M instead means that the error message comes only from the main script. The error message reported on the table is the one printed on the log file by M, since the one printed by S is equivalent.

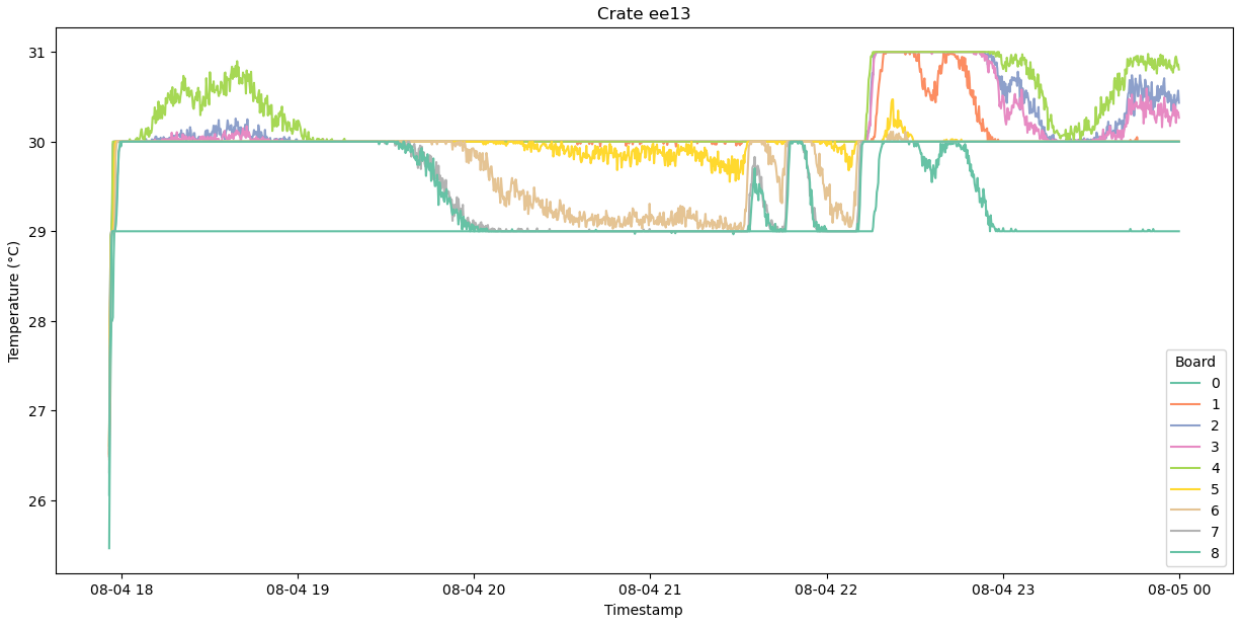

FIG. 5. Temperature time series of the crate EE13 during run 10235. It is possible to notice the highly populated horizontal lines that correspond to integer temperature values.

III. TEMPERATURE ANALYSIS AND CHARACTERIZATION

After acquiring and converting temperature data into a user-friendly format, the subsequent step involves data analysis. The primary objective is to define the temperature distribution

across the boards under normal operating conditions and contrast it with the distribution observed during abnormal temperature behaviour. For this purpose, we have chosen two specific runs - 10235 and 10265 (August 2023) - that have been processed and integrated into the ICARUS production database by exploiting the tools showcased in Sec. II.

A. Run 10235 analysis

After gaining access to the run data, very interesting details can be observed by looking at the time series visualization of the board temperatures within a given crate. As one can see in Fig. 5, the temperatures populate the horizontal lines that correspond exactly to integer values, while displaying some variability during shorter time intervals. This evidence raised some questions about the nature of the data itself since some fluctuations around an "equilibrium" temperature value should be expected in principle. Inspecting the registers

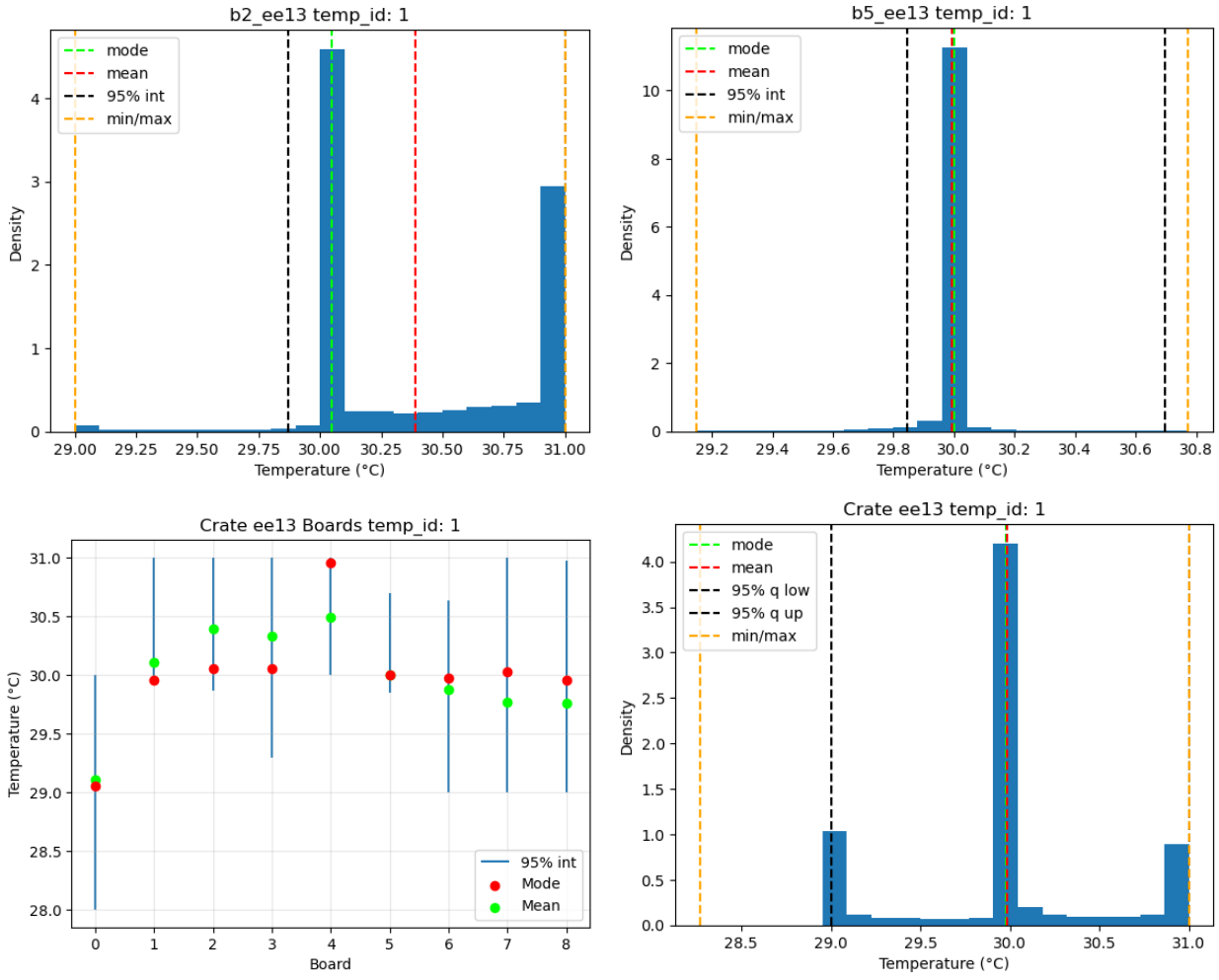

FIG. 6. *From top-left to bottom-right*: (1): temperature histogram of board 2 of crate EE13. (2): temperature histogram of board 5 of crate EE13. (3): temperature distributions of the boards inside crate EE13. (4) temperature histogram of crate EE13. All the plots contain Run 10235 data.

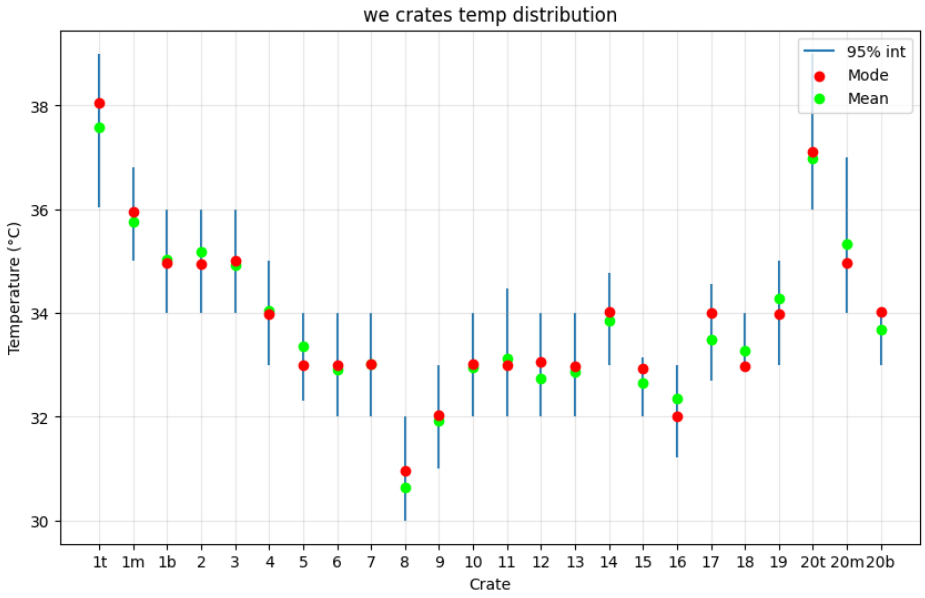

FIG. 7. Behaviour of the crate temperature distribution of the WE row during Run 10235. It is possible to notice the U-shaped behaviour due to the fact that the air conditioning outlets are located on the middle of the vessels main side, providing more cooling effect for the crates that lay in the middle of a crate row.

of the A2795 boards where the temperature sensors outputs are latched, it is possible to see that the temperature values are actually stored as 8-bit integers. However, non-integer values - with an unknown original precision - are present in the dataset, thus making us suppose that some sort of aggregation occurs at the board or at the `BoardReader` process level. A reasonable hypothesis is that the logging time - which corresponds to 15 s - is larger than the sensor sampling time, and thus if the temperature changes within a logging time window then the board or the process returns the average of the sampled (integer) temperatures. However, this aspect is not crucial for our work since we do not need a temperature sensitivity below 1°C for discriminating normal temperature conditions from abnormal ones: for this reason, no further investigation was needed. The temperature histograms for a single board inside a given crate (Fig. 6, (1)) show that the temperature distribution is in most cases bimodal and highly peaked on a integer value, as already observed. Less often, the distribution displays a single peak, as in Fig. 6 (2). If we compare the distributions of the board temperatures of all the boards inside a crate, as shown in Fig. 6 (3), it is possible to notice that they are heterogeneous, even if most of them share the same modal value and approximately the same 95% percentile interval. This results in crate temperature histograms, like the one in Fig. 6 (4), that display three peaks: the central one due to the contribution of the central peaks of the set of boards that share the same temperature behaviour, and the marginal ones that are due to the boards that are likely to be hotter or cooler than the others. However, the most interesting results come from comparing the crate temperature distributions for the crates belonging to the same row. Indeed, as one can see in Fig. 7, the distributions follow a U-shaped behaviour that develops along the row itself. This is due to the fact that the air conditioning system installed in the Far Detector mezzanine blows the fresh air directly in the middle of the two lateral corridors, that extend along the two longest sides, thus leaving the row endpoints much less exposed to the airflow (cf Fig. 1 (4) and Fig. 2, right). This

fact allows us to focus more on the temperature of the endpoint stacked crates - especially the top crates - since, if the temperature rises, they will be affected the most, being the crates that are on average sensibly hotter than the others.

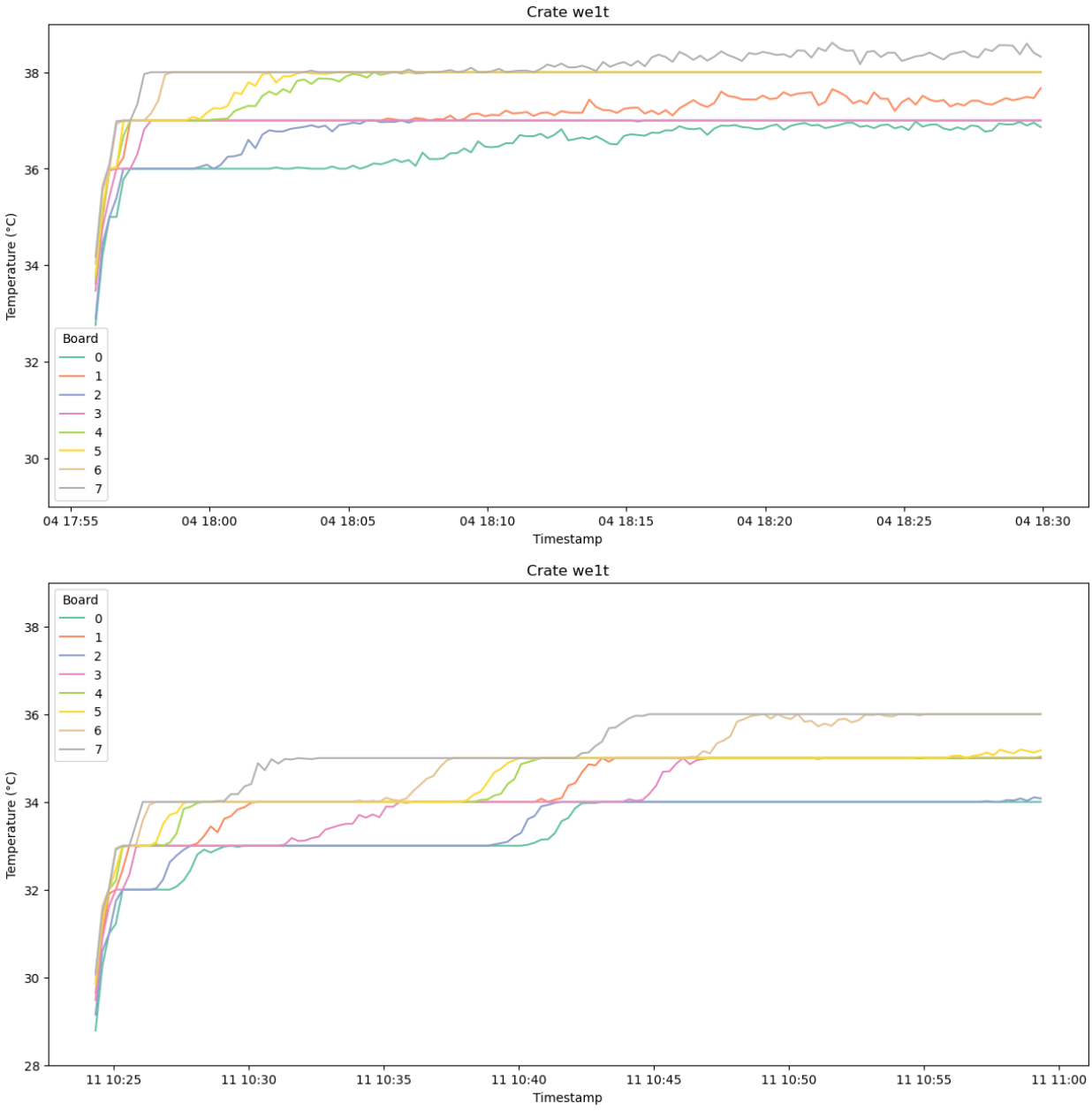

FIG. 8. *Above*: temperature time series for crate WE01T during run 10235. *Below*: temperature time series for crate WE01T during run 10265. It is worth noticing how differently the temperature behaves in the two cases in the considered time interval ($\sim$ 35 min), given that the data comes from similar situations (i.e. the beginning of the run), except for the absence of air conditioning for Run 10265.

B. Run 10265 analysis

Run 10265 has been launched on purpose without mezzanine air conditioning after a power outage, with the specific intent of simulating a situation in which the air conditioning fails causing the board temperatures to rise. Since the experiment was shut down for a while, the electronics was cold and thus the initial temperatures did not reflect the usual ones, which are of course higher. Nevertheless, this trial has proven valuable in determining the time interval needed for the temperature to rise by a specific magnitude, potentially breaching a safety threshold. The temperature time series is reported in Fig. 8, bottom: it is possible to see that for most boards the temperature takes about 10 minutes to increase by 2°C. The effects of the temperature rise are also visible from the histograms, which start to display a stair-like shape with more than two peaks.

C. Analysis results

From the analysis performed on the data coming from runs 10235 and 10265 we retrieve the following important results:

1. Each board inside a crate is characterized, most of the times, by a bimodal distribution with a dominant peak that defines a characteristic temperature.
2. The boards inside the same crate do not share a common distribution, but most of them display the same modal value and similar 95% percentile intervals. This means that the temperature distribution of a crate is characterized by a leading modal temperature with a spread given by the 95% percentile interval that ranges $(1 \div 2)$°C.
3. The crate temperature distributions follow a U-shaped behaviour when plotted against the longitudinal coordinate that spans the row which the crates belong to. This reflects the fact that the top stacked crates at the endpoints of each row are the hottest ones, with a characteristic temperatures of $(38 \div 39)$°C. Moreover, this means that we have to monitor those crates first, since they will be the most in danger in anomalous temperature conditions.
4. In case of an air conditioning failure we observed that the boards are, on average, subject to a temperature rise of ~ 2°C in a timespan of approximately 10 minutes.

IV. GRAFANA ALERTING SYSTEM

The results obtained from the data analysis have proven valuable in order to implement a temperature alerting system based on Grafana [3], which is an open-source analytics and interactive visualization web application. Grafana is currently running on the ICARUS online machines and its data source is the Graphite database that we mentioned earlier in Sec. II. This means that from Grafana one can access all the metrics that are pushed by the processes that read out the boards - not only the TPC boards - inside the Graphite database and build dashboards that help visualize the data or/and simple derived quantities like averages, maximum/minimum values etc. As we discussed before, the top stacked crates at the endpoints of each crate row are the hottest ones, so it is reasonable to set up alarms that focus on those crates, since they will be affected the most by an occurring temperature

FIG. 9. Sample image of the Grafana dashboard realized for monitoring the 8 top stacked crates. It is possible to notice the little hearts that, together with the green and red markers visible on the bottom-left corner, warn the user if the alarm associated with the panel is firing.

rise. To do so, it is convenient to build a dashboard with panels that collect the temperature time series for each board inside those crates: in this way, one can pin the 8 targeted crates panels at the top of the dashboard and keep an eye on the health status of their alarms (see Fig. 9). Indeed, there are currently 9 alerts set up on Grafana: 8 of them are temperature-monitoring-related rules associated to the 8 top crates and one is a NoData alert that notifies run crashes.

A. Temperature alert rules

The temperature alerts are structured in order to go through the following steps every 2 minutes:

1. Fetch the current temperature value for each board and check if the difference with the temperature value of 10 minutes back in time of the same board is above 1.5 °C.
2. Check if during the past 20 minutes the run number has not changed.
3. Fetch the last temperature value for each board and check if it is above 38 °C.

Then, an alarm is fired if the statement $(1 \wedge 2) \vee 3$ returns True. If the alarm fires then a Slack notification is sent to the `test_grafana_alerts` Slack channel. Conditions 1 and 3 are directly implemented in view of the results obtained from the data analysis. Condition 2 instead, is a caveat for avoiding the alarm to fire when a run is starting, since in that case it is expected that the temperatures increase, starting of course from a much lower temperature baseline.

B. NoData alert rule

While exploring the tools provided by Grafana, we noticed that a valid condition for implementing conditional actions based on temperature time series values is checking for the presence of NoData values. A NoData value is retrieved by Grafana whenever the data source stops pushing data: in our case, this happens when a run stops or crashes suddenly. For this reason, an alert for checking the presence of NoData values can be helpful to spot run crashes without the active supervision of the shifter. Moreover, just a single alert is enough since, if the run crashes, all the components that were included at the beginning of the run crash at the same time. Thus, we decided to link the NoData alert to the crate EE01T and set up it in order to check every minute if the last value coming from any board inside the crate retrieves NoData. A NoData value has to be persistent for at least 5 minutes in order to avoid firing an alarm if an occasional NoData value occurs. If the alarm fires, then a Slack notification is sent to the `test_nodata_alerts` channel.

V. ACCESSING AND USING THE DATA

In this section we are going to show how one can access the board temperature data inside the ICARUS production database in order to generate plots, perform further data analysis, etc. As we mentioned before, the production database relies on the ICARUS online machines, but it is also replicated on an offline database (`ifdb09`) to allow users to access it without passing through the online cluster gateway. A convenient and easy way to access the database - both online and offline - is working with Jupyter notebooks combined with the JupySQL [7] library, which allows using line-oriented or cell-oriented magic commands like `%sql` and `%%sql` respectively: the former allows the user to store the output of a SQL queries directly inside a Pandas Dataframe object while the latter enables the inline visualization of a SQL query output below the notebook cell. The combination of the two commands allows both to perform operations on the data and inspect the database according to the needs. Moreover, the user can set up a `connections.ini` file inside `~/.jupysql/` in order to load a default connection without having to specify the database coordinates inside each notebook file.

VI. CONCLUSIONS

In this work we have showcased our solutions for implementing an efficient and reliable system for monitoring the TPC readout boards temperature. Through a series of scripts, temperature data for a specific run is fetched directly from the metric log files - generated by the `BoardReader` process - and is pushed inside the ICARUS production database (`icarus_online_prd`) and its offline replica (`ifdb09`). The data coming from Runs 10235 and 10265 has been analyzed in order to provide a temperature characterization at a board, crate and row of crates level. Insights coming from the data analysis have been exploited in order to implement alerts on Grafana necessary to warn the shifters when the board temperatures display a pathological behaviour. Also a NoData alert has been implemented in order to detect run crashes efficiently. Future developments of this work may include improving the reliability and the organization of the code on one hand and extend the data analysis by including temperature data coming from the board power supplies on the other. This will allow studying the correlation between the temperatures in order to search for other

valuable insights.

BIBLIOGRAPHY

[1] ICARUS Collaboration. Icarus at the fermilab short-baseline neutrino program – initial operation, 2023.

[2] CAEN Group. A2795 liquid argon tpc readout board. https://www.caen.it/products/a2795/. Visited on 3 November 2023.

[3] Grafana Labs. Grafana. https://grafana.com/grafana/. Visited on 3 November 2023.

[4] Krishanu Majumdar and Konstantinos Mavrokoridis. Review of liquid argon detector technologies in the neutrino sector. *Applied Sciences*, 11:2455, 03 2021.

[5] Kurt Biery, Eric Flumerfelt, John Freeman, Wesley Ketchum, Gennadiy Lukhanin, Adam Lyon, Ron Rechenmacher, Ryan Rivera, Lorenzo Uplegger, and Margaret Votava. Flexible and scalable data-acquisition using the artdaq toolkit, 2018.

[6] Graphite. Graphite project. https://graphiteapp.org/#overview. Visited on 3 November 2023.

[7] Jupysql. https://jupysql.ploomber.io/en/latest/quick-start.html#. Visited on 3 November 2023.