The DEBS 2020 Grand Challenge

Vincenzo Gulisano
Chalmers University of Technology
Sweden
vincenzo.gulisano@chalmers.se

Daniel Jorde
Technical University of Munich
Germany
daniel.jorde@tum.de

Ruben Mayer
Technical University of Munich
Germany
ruben.mayer@tum.de

Hannaneh Najdataei
Chalmers University of Technology
Sweden
hannajd@chalmers.se

Dimitris Palyvos-Giannas
Chalmers University of Technology
Sweden
palyvos@chalmers.se

ABSTRACT

The ACM DEBS 2020 Grand Challenge is the tenth in a series of challenges which seek to provide a common ground and evaluation criteria for a competition aimed at both research and industrial event-based systems. The focus of the ACM DEBS 2020 Grand Challenge is on Non-Intrusive Load Monitoring (NILM). The goal of the challenge is to detect when appliances contributing to an aggregated stream of voltage and current readings from a smart meter are switched on or off. NILM is leveraged in many contexts, ranging from monitoring of energy consumption to home automation. This paper describes the specifics of the data streams provided in the challenge, as well as the benchmarking platform that supports the testing of the solutions submitted by the participants.

CCS CONCEPTS

• **General and reference** → **Performance**; • **Information systems** → **Data streams**;

KEYWORDS

Event processing, Data Streaming, NILM

ACM Reference Format:
Vincenzo Gulisano, Daniel Jorde, Ruben Mayer, Hannaneh Najdataei, and Dimitris Palyvos-Giannas. 2020. The DEBS 2020 Grand Challenge. In *The 14th ACM International Conference on Distributed and Event-based Systems (DEBS '20), July 13–17, 2020, Virtual Event, QC, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3401025.3402684

1 INTRODUCTION

The ACM DEBS 2020 Grand Challenge is the tenth in a series [2, 5–10, 12] of challenges that seek to provide a common ground and evaluation criteria for a competition aimed at both research and industrial event-based systems.

The ACM DEBS 2020 Grand Challenge focuses on Non-Intrusive Load Monitoring (NILM). The goal of the challenge is to detect when appliances contributing to an aggregated stream of voltage and current readings from a smart meter are switched on or off. NILM is leveraged in many contexts, ranging from monitoring of energy consumption to home automation. The ACM DEBS 2020 Grand Challenge evaluation platform is provided by the Distributed Computing and Systems (DCS) Research Group (Chalmers). Such platform is based on the evaluation platform previously provided by the Systems Engineering Group (TU Dresden).

This paper presents the data that was used in the challenge in Section 2 and the tasks that participants had to solve in Section 3. Section 3.3 introduces the evaluation criteria for the ACM DEBS 2020 Grand Challenge and Section 4 describes the platform and the benchmark implementation used for the evaluation.

DEBS '20, July 13–17, 2020, Virtual Event, QC, Canada

ACM ISBN 978-1-4503-8028-7/20/07.
https://doi.org/10.1145/3401025.3402684

2 DATA

The data provided for the challenge is a subset of the published BLOND dataset [11] which consists of high sampling rate energy measurements from a custom-built smart meter in an office building. The data represents the aggregate consumption from multiple offices and is enriched with the per-appliance consumption ground-truth to enable energy disaggregation. Each input tuple $\langle i, v, c \rangle$ represents an energy measurement, where attributes i, v and c represent the tuple sequence id, the voltage and the current, respectively.

In order to use these input tuples to perform NILM event detection, this data needs to be preprocessed in the participants' solution. Each input tuple is first aggregated using a tuple-based tumbling window W_1 of size 1000 to compute the active and reactive power features. The data is sampled 50000 times per second, hence, every period in the signal contains 1000 samples, based on the 50 Hz base frequency in Germany. The input features are computed per period as follows:
Active Power $P = \frac{\sum(v \times c)}{1000}$.
Apparent Power $S = voltage_{RMS} \times current_{RMS}$ with root-mean-square (RMS) values of voltage and current per period respectively.
Reactive Power $Q = \sqrt{S^2 - P^2}$.

3 THE GRAND CHALLENGE PROBLEM DEFINITION

This section introduces the task that participants of the ACM DEBS 2020 Grand Challenge had to solve. The goal is to detect when appliances contributing to an aggregated stream of voltage and current readings are switched on or off. The challenge solution

should produce a result every 1000 input tuples (i.e., each time window W_1 is full). Such result specifies whether an event has been detected for such window and the sequence number of the window W_1 for which the event is detected.

The schema of the output stream is thus: $\langle s, d, event_s \rangle$, where s is the window W_1 input tuple sequence id to which this output tuple refers to, d is a boolean attribute that specifies whether an event is detected or not and $event_s$ is the window W_1 sequence id of the detected event (if any).

In particular, in this year's challenge, we have introduced two different queries to be solved: Query 1 is the described event detection on a regular stream of input tuples, whereas in Query 2, the input stream could arrive out-of-order or even be incomplete. In the following, we describe both queries in detail.

3.1 Query 1

As described in Section 2, detecting the events involves computing active and reactive powers from the input tuples in window W_1. This results in a stream of features, more precisely, of active and reactive powers in the corresponding W_1 window instances.

The resulting stream of features is then processed based on the algorithm described by Barsim and Yang [1]. More concretely, a tuple-based window W_2 of varying size is maintained and each new pair of features (active and reactive power) are added to window W_2. The DBSCAN algorithm [3] is applied to the window, using a forward and a backward pass.

In the forward pass, first the event model constraints are checked. Then, the clustering loss is computed. If the model constraints are valid and the loss is below a given threshold, an event is detected. In this case, the backward pass is started. Else, the next tuple is added to window W_1 and the algorithm continues from the start.

In the backward pass (i.e., when an event is detected in the forward pass), the oldest data tuple is removed from W_2 in each iteration and the DBSCAN clustering algorithm is applied. This is repeated as long as the event that was detected previously in the forward pass is still detected. If it is not detected anymore, the backward pass is stopped and the resulting event is returned. By doing so, one gets more stable steady-state sections from the algorithm. If an event is not detected and window W_2 contains more than 100 elements, then W_2 is emptied.

3.2 Query 2

The second query is a variation of Query 1. More concretely, this query is expected to process an input stream that can contain both late (out-of-order) arrivals as well as missing tuples. Since the semantics of Query 2 are equal to those of Query 1, participants are expected to provide a solution that is able to trade-off timeliness and accuracy of results. Upon reception of an input tuple, the solution can produce an output (based only on the data observed so far) or decide to postpone the output and wait until late-arriving input tuples are received. In this second case, the solution can wait for possible late arrivals to provide an output that more accurately identifies the timestamp at which an event is detected (if any). In order to differentiate between missing tuples and late arrivals, we bounded the maximum delay of late tuples to 20 W_1 windows (20,000 time units). Each output tuple must be produced only once.

3.3 Evaluation

The overall final rank is calculated as the sum of all rankings for both queries. The solution with the lowest overall final rank wins the performance award. The paper review score is used in case of ties.

Query 1. Evaluation of Query 1 addresses two aspects: (1) correctness of results and (2) processing speed. The first is taken into account by comparing the results of a proposed solution with that of our baseline. Only solutions that produce correct results (i.e., that produce the same set of output tuples produced by our baseline and in the same order) are considered as valid. The second aspect is captured with multiple measures, the total run-time (rank_0) and the latency (rank_1). The specifics of the ranking for the processing speed and quality of results are defined as follows. The total run-time (rank_0) is the time span between the sending of the first input tuple and the reception of the result for the last input tuple — the lower the total run-time measure, the higher the position in the ranking. The latency (rank_1) is measured as the average time span between retrieving an input tuple and providing the corresponding output tuple — the lower the latency, the higher the position in the ranking.

Query 2. Evaluation of Query 2 addresses two aspects: (1) timeliness of produced results (rank_3) and (2) accuracy (rank_4).

We define t_{out} as the *expected* output tuple produced by a hypothetical baseline system that is fed all input data in order upon processing of an input tuple t_{in}. Further, we define t'_{in} as the latest tuple retrieved by a participant's system when the output tuple t'_{out} is produced. Now, t'_{in} is possibly larger than t_{in}, as the participants solution may buffer events to cope with out-of-order situations. The timeliness of each output tuple t'_{out} is then computed as:

$$\max(0.1 - \frac{(t'_{in}.i - t_{in}.i)}{10.0}).$$

The accuracy is computed for output tuples in which an event is detected by the hypothetical baseline solution as:

$$\max(0.1 - |\frac{(t'_{out}.event_s - t_{out}.event_s)}{10.0}|).$$

4 EVALUATION PLATFORM

As mentioned in the previous section, the task for this year's DEBS challenge is to detect when appliances contributing to an aggregated stream of voltage and current readings from a smart meter are switched on or off. Solutions can be implemented in any programming language. The communication between the solution and the evaluation platform is carried out through a simple REST-based protocol. The solution, i.e., the contestant's implementation code, makes GET requests to receive data in batches of 1000 energy measurements as a JSON object, it runs the detection and transmits the answer using a POST request. In order to assess the performance of the solutions, contestants are required to package their solutions in Docker containers, which communicate with an evaluation container via REST.

The docker-based setup allows running performance evaluations in two ways: (i) locally, where the contestants use their infrastructure to test and evaluate their system's performance, and (ii) through the official evaluation platform. Both evaluation settings utilize the same evaluation container that provides the data and checks the provided answers. The official platform provides

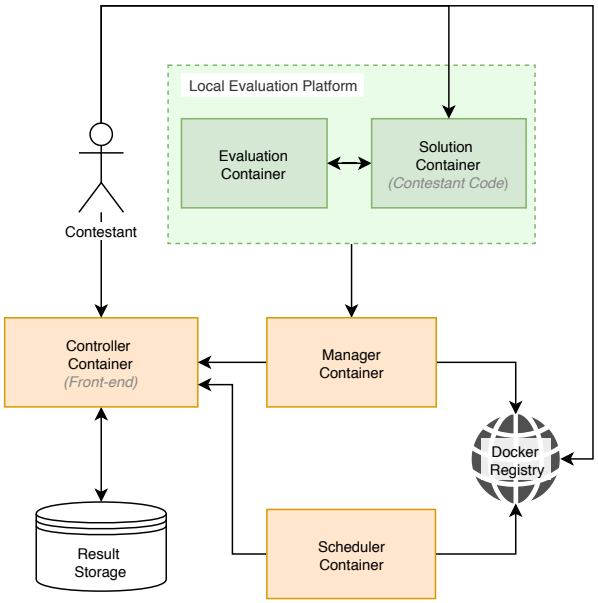

Figure 1: Architecture of the online evaluation platform.

additional functionality such as automatic re-evaluation, result persistence, and a web-based leader-board ranking the submissions of all contestants. To enable automatic re-evaluation when solutions are updated, contestants make their submissions available at a docker registry. The docker registry is periodically polled by the platform, and a new run is triggered whenever an updated image is identified. In the following, we briefly present the architecture of the online evaluation platform.

Architecture

The overall architecture of the online evaluation platform [4] is based on the principles of the HOBBIT's platform[1], extended and slightly redesigned to support generic interfaces such as REST. This new approach enables distributed but also local evaluation, allowing contestants to assess their system performance using their own infrastructure, without the need for modifications in their setup. Local evaluation requires two containers: the solution container and the evaluation container that checks the results of the solution. The online evaluation platform uses the same evaluation and solution containers, as well as some additional ones that provide services that drive the experiments in a distributed and automated manner, as depicted in Figure 1. The components are described below. They run on a server equipped with Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz with 2 sockets, 8 cores per socket, and 2 threads per core and 128GB of RAM. The evaluation platform is written in Python and uses frameworks such as Flask and existing infrastructure tooling such as Docker.

Controller. The controller can be envisioned as a mediator component for all services. It provides a REST API for collecting and presenting the evaluation results, and it also persists its state in a separate database container. The controller runs a web-server for displaying the results online as a ranked leader-board.

Manager. The manager component is responsible for triggering the performance evaluation by automatically launching and tearing down the solution and evaluation containers. It also collects the results of each benchmark and sends them to the controller for ranking and persistence.

Scheduler. The scheduler acts as a helper service. It queries the remote docker registry and checks for updated solution containers. In order to detect changes, the scheduler maintains the hashes of the container images and requests a re-evaluation if they differ from the latest ones in the registry. The re-evaluation is then performed by the manager component.

Performance Optimizations. In order to provide batches of data as quickly as possible to the solution code upon a GET request, the data is pre-loaded asynchronously in a batching manner from the files. The batches are then kept in memory until they are requested by the client.

[1]https://project-hobbit.eu/wp-content/uploads/2018/03/D2.2.2_Second_Version_of_the_HOBBIT_Platform.pdf

ACKNOWLEDGMENTS

This year's Grand Challenge is co-organized by the Distributed Computing and Systems (DCS) Group at Chalmers University (https://www.chalmers.se/en/) and the Middleware Systems Research Group at Technical University of Munich (TUM), that made available the evaluation server for ranking the participants' Grand Challenge solutions.

REFERENCES

[1] Karim Said Barsim and Bin Yang. 2016. Sequential clustering-based event detection for non-intrusive load monitoring. *Computer Science & Information Technology* 6 (2016), 77–85.
[2] Oleh Bodunov, Vincenzo Gulisano, Hannaneh Najdataei, Zbigniew Jerzak, André Martin, Pavel Smirnov, Martin Strohbach, and Holger Ziekow. 2019. The DEBS 2019 grand challenge. In *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems*. 205–208.
[3] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*. AAAI Press, 226–231.
[4] GitHub. 2020. DEBS 2020 Grand Challenge Evaluation Platform. https://github.com/dmpalyvos/debs-2020-challenge
[5] Vincenzo Gulisano, Zbigniew Jerzak, Roman Katerinenko, Martin Strohbach, and Holger Ziekow. 2017. The DEBS 2017 Grand Challenge. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems (DEBS '17)*. ACM, New York, NY, USA, 271–273. https://doi.org/10.1145/3093742.3096342
[6] Vincenzo Gulisano, Zbigniew Jerzak, Pavel Smirnov, Martin Strohbach, Holger Ziekow, and Dimitris Zissis. 2018. The DEBS 2018 Grand Challenge. In *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems, DEBS 2018, Hamilton, New Zealand, June 25-29, 2018*. 191–194. https://doi.org/10.1145/3210284.3220510
[7] Vincenzo Gulisano, Zbigniew Jerzak, Spyros Voulgaris, and Holger Ziekow. 2016. The DEBS 2016 Grand Challenge. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems (DEBS '16)*. ACM, New York, NY, USA, 289–292. https://doi.org/10.1145/2933267.2933519
[8] Zbigniew Jerzak, Thomas Heinze, Matthias Fehr, Daniel Gröber, Raik Hartung, and Nenad Stojanovic. 2012. The DEBS 2012 grand challenge. In *Proceedings of the Sixth ACM International Conference on Distributed Event-Based Systems, DEBS 2012, Berlin, Germany, July 16-20, 2012*, François Bry, Adrian Paschke, Patrick Th. Eugster, Christof Fetzer, and Andreas Behrend (Eds.). ACM, 393–398. https://doi.org/10.1145/2335484.2335536
[9] Zbigniew Jerzak and Holger Ziekow. 2014. The DEBS 2014 grand challenge. In *The 8th ACM International Conference on Distributed Event-Based Systems, DEBS '14, Mumbai, India, May 26-29, 2014*, Umesh Bellur and Ravi Kothari (Eds.). ACM, 266–269. https://doi.org/10.1145/2611286.2611333

[10] Zbigniew Jerzak and Holger Ziekow. 2015. The DEBS 2015 Grand Challenge. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, DEBS '15, Oslo, Norway, June 29 - July 3, 2015*, Frank Eliassen and Roman Vitenberg (Eds.). ACM, 266–268. https://doi.org/10.1145/2675743.2772598

[11] Thomas Kriechbaumer and Hans-Arno Jacobsen. 2018. BLOND, a building-level office environment dataset of typical electrical appliances. *Scientific data* 5 (2018), 180048.

[12] Christopher Mutschler, Holger Ziekow, and Zbigniew Jerzak. 2013. The DEBS 2013 grand challenge. In *The 7th ACM International Conference on Distributed Event-Based Systems, DEBS '13, Arlington, TX, USA - June 29 - July 03, 2013*, Sharma Chakravarthy, Susan Darling Urban, Peter Pietzuch, and Elke A. Rundensteiner (Eds.). ACM, 289–294. https://doi.org/10.1145/2488222.2488283