Dependable IoT Data Stream Processing for Monitoring and Control of Urban Infrastructures

Morgan K. Geldenhuys, Jonathan Will, Benjamin J. J. Pfister,
Martin Haug, Alexander Scharmann, and Lauritz Thamsen
Technische Universität Berlin, Germany, {firstname.lastname}@tu-berlin.de

***Abstract*—The Internet of Things describes a network of physical devices interacting and producing vast streams of sensor data. At present there are a number of general challenges which exist while developing solutions for use cases involving the monitoring and control of urban infrastructures. These include the need for a dependable method for extracting value from these high volume streams of time sensitive data which is adaptive to changing workloads. Low-latency access to the current state for live monitoring is a necessity as well as the ability to perform queries on historical data. At the same time, many design choices need to be made and the number of possible technology options available further adds to the complexity.**

In this paper we present a dependable IoT data processing platform for the monitoring and control of urban infrastructures. We define requirements in terms of dependability and then select a number of mature open-source technologies to match these requirements. We examine the disparate parts necessary for delivering a holistic overall architecture and describe the dataflows between each of these components. We likewise present generalizable methods for the enrichment and analysis of sensor data applicable across various application areas. We demonstrate the usefulness of this approach by providing an exemplary prototype platform executing on top of Kubernetes and evaluate the effectiveness of jobs processing sensor data in this environment.

***Index Terms*—internet of things, distributed stream processing, real-time analytics, sensor data enrichment, complex event processing**

I. Introduction

The Internet of Things (IoT) has emerged as an important technological paradigm whereby large numbers of ubiquitous devices are networked to enable the real-time monitoring and control of physical infrastructures across a wide range of application areas. IoT owes its existence to the convergence of a number of key enabling technologies such as ubiquitous computing, commodity sensors and micro-controllers, machine-learning, and real-time analytics [1]. Important areas of application include the management of critical infrastructures such as human health-care, transportation systems, electrical generation, natural disaster prediction, and telecommunications, to name but a few [2]–[5]. With the number of sensor devices only projected to increase, i.e. from 10 billion in 2021 to more than 25 billion devices in 2030[1], it follows then that data volumes will likewise reach ever great heights in the years to come. Therefore, in order to meet processing demands a dependable IoT processing platform is necessary to ensure results are available when time-sensitive decisions need to be made as processing loads invariably increase over time.

However, building such a processing platform for geo-distributed urban infrastructures is not an easy task. There are a number of requirements, design choices, and technological options which need to be considered. Not only should the platform provide fast access to results describing the current state of the infrastructure to enable live monitoring, but also allow for the storage and analysis of historical data to identify trends over longer periods of time. Results should be presented in a format which is understandable to both the user and analysis jobs which at the same time ensures low-latency transmission of sensor data in an environment where network resources are limited. Likewise, there are a multitude of technology options from which to choose from for the fulfilment of individual use cases. This logically begs the question: which set of systems should be chosen so that when they are composed together they will produce a dependable solution?

Selecting the right technologies is not the only problem, such a platform consisting of complex inter-dependencies will likewise be difficult for operators of the platform to deploy and maintain. A number of solutions have been proposed in this area, however they tend to be specific to application domains not generalizable to urban infrastructures [6]–[9] or do not provide data about the scalability and/or reliability of their platform [10], [11]. Likewise, the number of commercially available end-to-end IoT solutions for the processing of large data streams is growing[2,3,4], however, these tend to be proprietary with little regard for open standards as vendors compete in the already lucrative IoT market. Additionally, such solutions are not feasible when the requirement for on-premise operations exist due to privacy and/or data protection issues.

In this paper we present a dependable data processing platform for IoT sensor streams. This open source platform is targeted towards the monitoring and control of urban infrastructures. We investigate and define the requirements for creating such a platform in the context of dependability with specific focus on availability, reliability, maintainability, and security. We propose an architecture consisting of a number of interconnected sub-systems which performs all necessary functions expected of a IoT data processing platform as well

[1]https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/, Accessed: Jul 2021

[2]https://cloud.google.com/solutions/iot, Accessed: Jul 2021

[3]https://aws.amazon.com/iot/, Accessed: Jul 2021

[4]https://www.ibm.com/cloud/internet-of-things, Accessed: Jul 2021

as ensuring that it is fast to deploy and easy to maintain. Likewise, we present two stream processing jobs essential to performing enrichment of sensor values as well as analysis as a generalizable use case for geo-distributed urban infrastructures. We perform experiments to evaluate the scalability and reliability of this setup and present the results. Our evaluation reveals our analysis platform architectures exhibits near-linear scaling capabilities for realistic example big data workloads.

The remainder of the paper is structured as follows: Section II presents an investigation of the requirements in the context of dependability; Section III presents the anatomy of the IoT data processing platform; Section IV details two stream processing jobs which are generalizable across most IoT monitoring and control use cases; Section V describes our evaluation through experiments; Section VI the related work; and Section VII summarizes our findings.

II. Platform Requirements

In this section we will discuss the requirements of a dependable IoT data processing platform. The function of such a platform is to consume the incoming IoT sensor data in order to extract valuable results by cleaning, filtering, enriching, aggregating, and analysing the time-sensitive data streams. Importantly, stream processing is concerned with providing high rates of throughput, low latency processing, and fault tolerant execution. Therefore, it is commonplace for constraints to exist which dictate the minimum performance requirements. For a platform of this type to be dependable, it should fulfil the following requirements.

- The platform should deliver the service that is required considering the presence of any performance constraints and adapt to any workload changes. For this to be true, components of the system need to be scalable.
- The platform should be able to continue operating in the presence of partial failures. Therefore, it is important that components are fault tolerant and able to recover automatically when failures occur.
- Distributed systems are notoriously complex and composing multiple of these systems together only exacerbates the problem. Therefore, the platform should abstract away this complexity from users making it relatively straightforward to use and maintain.
- The platform should be secured against malicious threats. Therefore, all externally facing interfaces should implement strict security measures and the number of these should be minimized to reduce the attack surface.

III. Platform Anatomy

Following on from the requirements presented in the previous section, here we detail our proposal for a dependable IoT data processing platform. We compose the platform from a number of key sub-systems which are interconnected and pass sensor streaming data between them. A graphical representation of this platform can be seen in Fig. 1. Data is generated from sensors placed in the field and flows through the platform where results are ultimately consumed by the monitoring system. The platform allows for both the consumption of live streaming data which is vital when considering the current state monitoring of a physical infrastructure, as well as, the capability to perform precision queries of historical data. Importantly, all components should provide the ability to secure outward facing communication channels to ensure the confidentiality and integrity of data. All subsystem recommendations mentioned in this section provide this functionality through encryption. Next we will describe the individual components of this platform and how they are intended to interact with one another.

A. Sensors and Gateways

Data is generated from sensors which are in turn connected to devices. Multiple sensors can be connected to the same device and each measures a specific physical quality, i.e. light intensity, distance, temperature, acceleration, etc. Oftentimes the devices are low powered microcontrollers with limited processing capacity. These devices will transmit the data to a gateway which acts as a bridge between the data generators and data processing environment. As sensor devices are generally remotely placed, they often use a variety of wireless protocols such as LoRa[5] or Zigbee[6] to transmit data and therefore require some intermediate infrastructure to pass data on to a TCP/IP based network. Examples of such infrastructures include ChirpStack[7] and The Things Network[8]. Regarding data formatting, it is important to reduce the amount of data which is transferred over these low bandwidth network. Therefore, the sensor network should transmit only the sensor ID, timestamp, and value while relying on the analytics platform to perform any enrichment and/or calibration of the data. A specification such as SenML should be used across the whole infrastructure for the encoding of sensor measurements and device parameters used in enrichment. For use cases where control functionality is necessary, devices are able to accept commands from some human-machine interface.

B. Distributed Message Streaming Platform

At the heart of the IoT data stream processing platform is a scalable message broker. This distributed middleware component allows for data to be published to and consumed from messaging queues. To ensure a high level of performance is maintained at any scale, all data should be routed through this platform as it decouples components from each other where slow sinks are a regular limiting factor. Therefore, instead of a pipeline of systems, a more star-like architecture is adopted with the message broker at the center. To ensure that this component does not itself become a performance bottleneck, it must exhibit two key characteristics: (1) the ability to replicate copies of the message queues across multiple brokers thus ensuring failure tolerance, and (2) the ability to partition individual message queues so that multiple sinks and

[5]https://lora-alliance.org/, Accessed: Jul 2021
[6]https://zigbeealliance.org/about/, Accessed: Jul 2021
[7]https://www.chirpstack.io/, Accessed: Jul 2021
[8]https://www.thethingsnetwork.org/, Accessed: Jul 2021

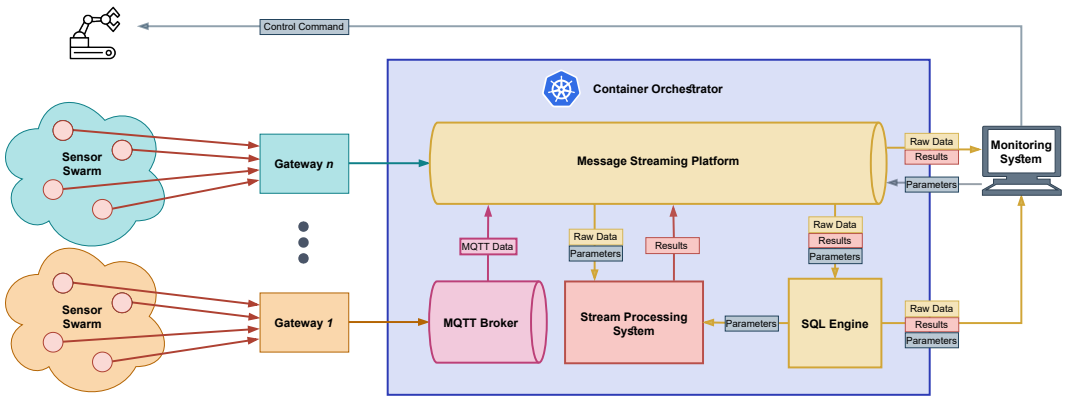


Fig. 1: Dependable IoT Data Processing Platform Architecture.

sources can consume/publish events in parallel. An example of such a system would be Apache Kafka [12]. Importantly, a system like Kafka can be configured to store individual message queues indefinitely and as such essentially becomes a permanent datastore. This setup is inherently scalable as well as fault tolerant. However, traditional database systems have the advantage of being able to perform intricate queries over the data which is normally not possible with these types of systems. Kafka, however, when combined with a SQL engine like ksqlDB[9] is able to provide this functionality. KsqlDB is a database which operates on top of Kafka which is purpose-built for stream processing use cases.

C. MQTT Broker

This optional component can be deployed if support for the Message Queuing Telemetry Transport (MQTT) protocol is required. As systems like Apache Kafka do not support MQTT, we can instantiate an intermediary broker as a target for MQTT data producers. RabbitMQ[10] is an example of such a system. Messages are produced from the MQTT gateway into this broker after which they are forwarded to the distributed message streaming platform where they are be processed as per normal.

D. Distributed Stream Processing System

DSP systems are becoming an increasingly essential part of any data processing environment. It is here where messages must traverse a graph of streaming operators to allow for the extraction of results which are at their most valuable at the time of data arrival. Such systems allow users to define streaming jobs and run them upon a cluster of commodity nodes which is able to scale out to process any potential workload. It is here where user defined jobs are executed. The raw sensor data is consumed from the message streaming platform after which it is enriched if necessary and any analysis performed. Results are outputted directly to the message streaming platform in order to bypass any slow sinks. Example systems include Apache Spark Streaming [13] and Flink [14]. The DSP system on startup will load any existing parameters from the SQL engine and store them in an in-memory data structure to be used during data enrichment and/or analysis processes. This mechanism will be further discussed in the subsequent section. It also provides a mechanism for users to update parameters during runtime execution. The *monitoring system* publishes the new parameters to the message streaming platform which in turn is consumed by the DSP and the local in-memory parameter cache updated. For the redundant storage of models and checkpoints, an external service such as Ceph [15] or HDFS [16] can be used.

E. Monitoring System

Users require a mechanism for interacting with the IoT platform where they can observe the current status of the system as well as the results of any analysis which may be performed. It is here where Geographic Information Systems (GIS), for example, are useful in the visualization of spacial and geographic data [17]. By integrating both message streaming and SQL clients, the monitoring system is able to consume the enriched IoT data to have an overview of the current state and perform intricate queries on the data over longer time periods. From here users have access to information necessary to make informed decisions about the infrastructures they are tasked with managing. Additionally, certain decision making can be automated to allow for the existence of self-optimizing processes. Although not reflected in Fig. 1, monitoring of the compute resources is likewise an important function. For this, a good solution is the deployment of a Prometheus[11] time series database as part of the platform which periodically scraps metrics from the underlying cluster nodes as well as sub-systems. From here, a data visualization frontend like Grafana[12] can be used to analyze performance.

F. Resource Management System

Resource management systems enable the automatic deployment, scaling, and management of containerized applications.

[9]https://ksqldb.io/, Accessed: Jul 2021
[10]https://www.rabbitmq.com/, Accessed: Jul 2021
[11]https://prometheus.io/, Accessed: Jul 2021
[12]https://grafana.com/, Accessed: Jul 2021

When a system like Kubernetes [18] is combined with a package manager such as Helm[13] we are able to use infrastructure-as-code processes to efficiently define, install, and upgrade a complex graph of interdependent sub-systems. These definitions can then be shared and updated with minimal effort without having to setup the individual systems manually. It thus reduces the need for expert knowledge on each individual sub-system and greatly reduces the lead time to deployment. Traditionally when deploying systems with a master-worker architecture on bare-metal, redundant nodes are required so that in the event of failure they can take over. However, the number of backups is finite and therefore can only tolerate a certain number of failures before the job ultimately stops processing. With Kubernetes and Flink native[14], for instance, where the stream processing system can communicate directly with the resource manager, it can now withstand an infinite number of failures by requesting new containers on demand. For the purposes of this paper we have created a github repository containing a helm chart for the infrastructure we describe in this section[15].

IV. General IoT Jobs

In this section we define two generalizable DSP jobs for the monitoring and control of urban infrastructures. These jobs rely on the architecture as described in the previous section to be in place. The first is the enrichment of IoT data which is necessary when using low bandwidth wireless infrastructures, thus ensuring low-latency processing. The second type of job uses complex event processing to allow, for instance, analyzing sensor measurements across wide geographical areas.

A. IoT Data Enrichment Job

Server side data enrichment is a key function of any stream processing platform when trying to optimize network resource utilization. This subsection describes how data transmitted using minimized formats can be enriched with information needed by the user in order to obtain an overview of the current state of the infrastructure as well as data analytics.

Inbound sensor data enters the stream processing system to be available for live monitoring and analytics. However, IoT sensors may transmit a variety of formats and data fields. For instance, a system might have a mix of IoT devices with digital and analog temperature sensors that can either directly emit a temperature or merely the sensor's voltage reading. The incoming data may also not be sufficient for monitoring or analysis, e.g. containing only an ID instead of the geographical latitude / longitude corresponding to this ID. Now, instead of having to deal with multiple formats in the analytics application and having to fetch missing data from a database, we perform a *data enrichment* job first. In this job, the stream processing system digests all inbound sensor data to transform it into a single, standardized format and annotates it with all information that may be needed for further consumption. These enriched packets are emitted back to the message streaming platform in a separate topic that analytics tasks can subscribe to.

Bandwidth conservation is important for low-power IoT devices to save energy. One way to achieve it, is to omit redundant data that can later be re-added to the datapoint. Additionally one can use a minimal data notation for transmission from the IoT device to the stream processing system, as defined by e.g. the *Sensor Measurement Lists*[16] (SenML). The following exemplifies sensor data in SenML notation.

```
{"n" : "70B3D5499073C7C0-temp",
 "t" : 1625222417,
 "v" : 4.2}
```

The first step to enriching this data is to look up the meaning of each key in a lookup table. We can deduct that in this example, we see a temperature measurement from an analog sensor with the name "70B3D5499073C7C0-temp", taken at Unix time 1625222417 and valuing at 4.2 Volts. After calibration, the enrichment job can assign a temperature to the voltage measurement, e.g. 23°C for 4.2 volts. Next, we can look up the geolocation of "70B3D5499073C7C0-temp" from a database and append it to the data point before ingesting it back into the message streaming system. The fully enriched data point might now look as follows:

```
{"latitude" : 52.51017262863814,
 "longitude": 13.322876673244508,
 "time" : 1625222417,
 "temperature" : 23}
```

One example application in the real world that makes use of IoT data enrichment is predictive pumping of water in urban sewage systems. Decisions here are based on continuous streams of data from geo-distributed sensors measuring suction chamber levels of water pumps on one hand and expected precipitation in an area of a city on the other hand. In this example, sending the data in a reduced format mitigates the load on the networking of the IoT devices attached to the sensors, while the enriched data retains full data interpretability by the analytics jobs that influence the pump controls.

To summarize, the enrichment job enables platform designers to appropriately separate data ingestion and formatting concerns from the actual analysis [19].

B. IoT Analytics Job

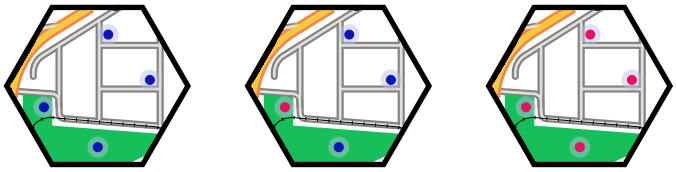
Once all data arrived in the central message streaming platform and was transformed to a standardized format, it has to be analyzed to extract insight. For this, we use *Complex Event Processing* (CEP). With this technique, the user specifies *event patterns* for data streams. Patterns allow to reason about a sequence of data packets in the stream and their relations, e.g. whether a value increases or exceeds a threshold multiple

[13]https://helm.sh/, Accessed Jun 2021

[14]https://ci.apache.org/projects/flink/flink-docs-release-1.13/docs/deployment/resource-providers/native_kubernetes/, Accessed: Jul 2021

[15]https://github.com/morgel/tdis21_dsp_platform, Accessed: Jul 2021

[16]https://www.iana.org/assignments/senml/senml.xhtml, Accessed: Jul 2021



(a) All sensors reporting within rolling average

(b) Anomaly type 1: One sensor anomalous

(c) Anomaly type 2: All sensors suddenly anomalous

Fig. 2: Proximity-based CEP uses a band around the rolling averages of sensors (dots) in a geographic cell. Red coloring indicates values outside of that band.

times within some time frame. A typical CEP pipeline would proceed as follows:

1) Event patterns are applied to an inbound enriched data stream, creating a new stream of possible events.
2) For each match an event is emitted to the output stream.
3) These event streams can then be transformed to alert events to be written back into the message streaming platform for further usage.

Buchmann et al. [20] state the two advantages that make CEP a good fit for a stream processing: Firstly, it separates analysis from the inbound data source and the outbound alert sink, thus offe nbring flexibility with regards to changes of the data producers in the urban infrastructure. Secondly, expressive event patterns make them especially suited to analyzing streams of data as opposed to looking at single data packets in an isolated fashion. CEP makes emitting alerts based on repeated deviation from a given value easy. However, many real-life systems do not have a single "normal" value to look out for. We examine this, using the example of sensors in street drainage with the task of reporting clogged inlets. The sensors observe water levels inside the inlet. The reference value depends on weather and other events. Assuming that most inlets are okay, we can obtain an estimate for the normal sensor value by averaging over all sensors in an area. The allowable sensor values can then only deviate by some fixed amount from that average, otherwise, an alarm is emitted.

We minimize the impacts of a faulty inlet on the reference value by calculating it as a rolling average. This offers two advantages: First, when a single anomaly starts, the system does not suddenly become less sensitive because that anomaly was already fully entered into the average. Second, many systems might consider sudden changes of a value across a set of neighbored sensors of interest as well. The rolling average will not fully update to the new, anomalous average and thus, all observed sensors are now outside of the range of values around the rolling average. Fig. 2 illustrates the concept. We use Uber's H3 library[17] to partition our sensors into hexagonal cells by location. We emit an alert if a sensor reports a water level outside the range around the rolling average for its cell twice or more within a short time frame.

[17]https://h3geo.org/, Accessed: Jul 2021

Another use case example for complex event processing is predictive maintenance in railway systems, in particular power transmission through pantograph-catenary systems. Complex models that take into account data from various sensors on the pantograph and cameras on the roof are used to assess the state of the system [21], [22]. Here, CEP enables live edge processing of data from the sensors in combination with recognized patterns from the video feed to detect anomalies[18].

V. Evaluation

In this section we evaluate the proposed dependable IoT data processing platform as well as both enrichment and analytical jobs through experiments designed to test scalability and reliability under different loads and cluster sizes.

A. Experiment Setup

Resource	Details
OS	Ubuntu 18.04.3
CPU	Quadcore Intel Xeon CPU E3-1230 V2 3.30GHz
Memory	16 GB RAM
Storage	3TB RAID0 (3x1TB disks, linux software RAID)
Network	1 GBit Ethernet NIC
Software	Java v1.11, Flink v1.12, Kafka v2.6, ksqlDB v6.1, RabbitMQ v3.8, ZooKeeper v3.6, Docker v19.3, Kubernetes v1.18, HDFS v2.8,, Prometheus v2.25

TABLE I: Cluster Specifications

Our experimental setup consisted of a 50-node co-located Kubernetes and HDFS cluster. Node specifications and software versions are summarized in Table I. A single switch connected all nodes. Each experiment consisted of a Kubernetes namespace containing: an Apache Kafka cluster of size 3 with all topics configured to have 8 partitions and a replication factor of 3; an Apache Zookeeper cluster of size 3; a single ksqlDB server; a single RabbitMQ server; an Apache Flink native session cluster[19]; and a single Prometheus[20] time series database was used for the gathering of metrics. We provide a Github repository[21] containing the helm chart for our cluster setup as well as the Flink enrichment and analytics jobs. Experiments were conducted 5 times with the median selected for our results.

B. Scalability Experiment

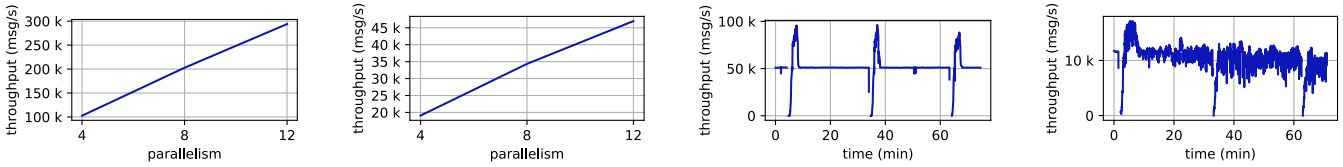
To test the scalability of the IoT data processing platform, we measured the maximum throughput for a data enrichment and an analytics job at varying cluster sizes. Because stream processing systems will process as many messages as possible using the resources available, the maximum processing capacity can be deduced by ensuring the messaging platform contains more messages than can be processed at a given time. Utilizing this fact, we generated enough messages to

[18]https://pantohealth.com/pantosys/, Accessed: Jul 2021

[19]https://ci.apache.org/projects/flink/flink-docs-stable/deployment/resource-providers/native_kubernetes.html, Accessed: Jul 2021

[20]https://prometheus.io, Accessed: Jul 2021

[21]https://github.com/dosgroup/tdis21_dsp_platform



(a) Scalability: Enrichment Job (b) Scalability: Analytics Job (c) Reliability: Enrichment Job (d) Reliability: Analytics Job

Fig. 3: Results of scalability and reliability experiments.

Kafka to sustain the maximum processing capacity for at least 20 minutes at each cluster size. We then recorded the mean throughput for the different cluster sizes across multiple runs.

From the results of the scalability experiments shown in Fig. 3a and Fig. 3b, we see that each job is able to process a different number of maximum messages at each cluster size. However, increasing the cluster size for both jobs resulted in a near-linear increase in processing capacity. We can furthermore deduce the amount of resources needed to provide dependable service. For example, with a cluster size of four, the data enrichment job can dependably process 50,000 messages per second, allowing for additional processing capacity in the event of failure.

C. Reliability Experiment

To test the reliability of the platform, we ran experiments where failures were injected into both the data enrichment and analytics jobs. Chaos mesh[22] was used to inject 3 failures every 30 minutes which caused a random task manager container to fail and the jobs to restart. We then measured the mean time to recover. Flink was configured to use a checkpoint interval of 10 seconds, a timeout interval of 50 seconds, and to use *exactly-once* fault tolerance guarantees. Workloads which oscillated close to 50K and 10K messages per second were used for the enrichment and analytics jobs respectively. A parallelism of 4 was used for both jobs resulting in an average utilization rate of 50% considering maximum processing capacities established in the scalability experiment.

Fig. 3c and Fig. 3d shows the input throughput over time for these experiments. Time taken to restart the job after failures were detected averaged 10 seconds across all jobs. The average time taken to process the backlog of accumulated events while the system was unavailable, i.e. the consumer lag, was 194 and 243 seconds for the enrichment and analytics jobs respectively. In all cases the jobs were able to recover and catch up to processing events at the latest timestamp.

D. IoT Analytics Experiment

For an evaluation of the alerts emitted by the proximity-based CEP job, we created a simulation with six geographical clusters (cells) of sensors, each containing between one and fifty sensors. We then simulated their measurements with values fluctuating around some mean per cell (with noise) and injected anomalies of magnitudes the job should emit alerts for for one, multiple, or all sensors in a cell.

[22]https://chaos-mesh.org/, Accessed Jul 2021

	Anomalous readings	Normal readings
Associated with alert	34	1689.8
No associated alert	0	27.2

TABLE II: Results of the location-aware CEP analysis

During five runs of our evaluation parcours, we produced three separate anomalous situations in different cells with a total of 12 sensors ever reporting values deviating from the mean. Table II shows that our system associated alarms to all anomalous readings as well as to 27.2 normal readings (1.6% false positive rate). The compromise between false positives and false negatives can be readjusted by choosing different parameters for the allowable deviation from the mean.

VI. Related Work

When looking at the current state of research one can find several papers adapting a similar architecture, or parts of it, for solving problems comparable with the ones introduced here. The paper of Sahal et al. [23] examines existing big data technologies for predictive maintenance use cases in the industry 4.0. Ye at al. [7] utilizes the scalability of Flink and sliding window partitioning to detect anomalies in hydrologic time series. A different application is presented in Difallah et al. [8] where Apache Storm is used to detect anomalies in a water distribution network. More work on the here proposed platform can be found in an earlier work of our research group in Lorenz et al. [19] Nasiri et al. [24] and Hesse and Lorenz [25] study capabilities and architecture of data stream processors. On the topic of message brokers, Moskvicheva and Dolgachev [26] compare RabbitMQ and Apache Kafka as two popular open source options. Other platform proposals can be found in Huru et al. [11] and Malek et al. [10] who developed general purpose platforms for distributed, scalable, and fault tolerant IoT data stream processing. No data about scalability or reliability of these approaches were given though. The paper of Zeuch et al. [6] and its fog computing approach is focused on future special cases where centralised processing might not be a viable solution. Our previous work in the area of dependable stream processing includes optimizing the fault tolerance mechanisms of DSP systems to ensure compliance with user-defined Quality of Service constraints for both performance and availability [27], [28]. Additionally, our work also explores the area of dependable data analytics platforms specific to water infrastructure monitoring [29], [30].

VII. Conclusion

In this paper we presented a dependable IoT data processing platform used for the monitoring and control of urban infrastructures. The goal of such a platform is to provide near real-time monitoring of the current state of the infrastructure as well as analytics for geo-distributed use cases such as predictive maintenance. The platform is intended to operate in the cloud or locally in a fog-based environment with the ability to adapt to any changes in the workload over time. We aimed to present an architecture which is not overly complex and composed of mature open-source software solutions. In the future we wish to perform further testing of this platform and jobs at greater scales using real world data streams.

Acknowledgment

This work has been supported through grants by the German Ministry for Education and Research (BMBF) as WaterGridSense 4.0 (funding mark 02WIK1475D) as well as OPTIMA which is co-financed by the European Regional Development Fund. We also thank Felix Lorenz for his contributions to this research and all the partners involved in the two funded projects acknowledged here.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *ArXiv*, vol. abs/1207.0203, 2013.

[2] D. Georgakopoulos, P. Jayaraman, M. Fazia, M. Villari, and R. Ranjan, "Internet of things and edge cloud computing roadmap for manufacturing," *IEEE Cloud Computing*, vol. 3, pp. 66–73, 2016.

[3] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami, "An information framework for creating a smart city through internet of things," *IEEE Internet of Things Journal*, vol. 1, pp. 112–121, 2014.

[4] B. Cheng, S. Longo, F. Cirillo, M. Bauer, and E. Kovacs, "Building a big data platform for smart cities: Experience and lessons from santander," *2015 IEEE International Congress on Big Data*, pp. 592–599, 2015.

[5] M. Lom, O. Pribyl, and M. Svitek, "Industry 4.0 as a part of smart cities," *2016 Smart Cities Symposium Prague (SCSP)*, pp. 1–6, 2016.

[6] S. Zeuch, A. Chaudhary, B. D. Monte, H. Gavriilidis, D. Giouroukis, P. M. Grulich, S. Bress, J. Traub, and V. Markl, "The nebulastream platform: Data and application management for the internet of things," 2020.

[7] F. Ye, Z. Liu, Q. Liu, and Z. Wang, "Hydrologic Time Series Anomaly Detection Based on Flink," *Mathematical Problems in Engineering*, vol. 2020, p. e3187697, May 2020, publisher: Hindawi. [Online]. Available: https://www.hindawi.com/journals/mpe/2020/3187697/

[8] D. E. Difallah, P. Cudré-Mauroux, and S. A. McKenna, "Scalable Anomaly Detection for Smart City Infrastructure Networks," *IEEE Internet Computing*, vol. 17, no. 6, pp. 39–47, Nov. 2013, conference Name: IEEE Internet Computing.

[9] S. Kolozali, M. Bermudez-Edo, D. Puschmann, F. Ganz, and P. Barnaghi, "A Knowledge-Based Approach for Real-Time IoT Data Stream Annotation and Processing," in *2014 IEEE International Conference on Internet of Things(iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)*. Taipei, Taiwan: IEEE, Sep. 2014, pp. 215–222. [Online]. Available: http://ieeexplore.ieee.org/document/7059664/

[10] Y. N. Malek, A. Kharbouch, H. E. Khoukhi, M. Bakhouya, V. D. Florio, D. E. Ouadghiri, S. Latre, and C. Blondia, "On the use of IoT and Big Data Technologies for Real-time Monitoring and Data Processing," *Procedia Computer Science*, vol. 113, pp. 429–434, Jan. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050917316903

[11] D. Huru, C. Leordeanu, E. Apostol, and V. Cristea, "BigClue: Towards a generic IoT cross-domain data processing platform," in *2018 IEEE 14th International Conference on Intelligent Computer Communication and Processing (ICCP)*. Cluj-Napoca: IEEE, Sep. 2018, pp. 427–434. [Online]. Available: https://ieeexplore.ieee.org/document/8516597/

[12] M. Sax, "Apache kafka," in *Encyclopedia of Big Data Technologies*, 2019.

[13] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. USA: USENIX Association, 2010, p. 10.

[14] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink™: Stream and batch processing in a single engine," *IEEE Data Eng. Bull.*, vol. 38, pp. 28–38, 2015.

[15] S. Weil, S. Brandt, E. L. Miller, D. Long, and C. Maltzahn, "Ceph: a scalable, high-performance distributed file system," in *OSDI '06*, 2006.

[16] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–10, 2010.

[17] P. Burrough and R. McDonnell, "Principles of geographical information systems," 1998.

[18] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," *Proceedings of the Tenth European Conference on Computer Systems*, 2015.

[19] F. Lorenz, M. Geldenhuys, H. Sommer, F. Jakobs, C. Lüring, V. Skwarek, I. Behnke, and L. Thamsen, "A scalable and dependable data analytics platform for water infrastructure monitoring," *arXiv preprint arXiv:2012.00400*, 2020.

[20] A. Buchmann and B. Koldehofe, "Complex event processing," *Information Technology*, vol. 51, no. 5, pp. 241–242, 2009.

[21] S. Bruni, J. Ambrosio, A. Carnicero, Y. H. Cho, L. Finner, M. Ikeda, S. Y. Kwon, J.-P. Massat, S. Stichel, M. Tur *et al.*, "The results of the pantograph–catenary interaction benchmark," *Vehicle System Dynamics*, vol. 53, no. 3, pp. 412–435, 2015.

[22] L. Finner, G. Poetsch, B. Sarnes, and M. Kolbe, "Program for catenary–pantograph analysis, prosa statement of methods and validation according en 50318," *Vehicle System Dynamics*, vol. 53, no. 3, pp. 305–313, 2015.

[23] R. Sahal, J. G. Breslin, and M. I. Ali, "Big data and stream processing platforms for Industry 4.0 requirements mapping for a predictive maintenance use case," *Journal of Manufacturing Systems*, vol. 54, pp. 138–151, Jan. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0278612519300937

[24] H. Nasiri, S. Nasehi, and M. Goudarzi, "Evaluation of distributed stream processing frameworks for IoT applications in Smart Cities," *Journal of Big Data*, vol. 6, no. 1, p. 52, Dec. 2019. [Online]. Available: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0215-2

[25] G. Hesse and M. Lorenz, "Conceptual Survey on Data Stream Processing Systems," in *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, Dec. 2015, pp. 797–802, iSSN: 1521-9097.

[26] K. Moskvicheva and M. Dolgachev, "Industry Paper: Kafka versus RabbitMQ. A comparative study of two industry reference publish/subscribe implementations," *Youth Science Forum Journal*, vol. 1, no. 4, pp. 3–17, Oct. 2020. [Online]. Available: http://forummn.ru/article-1.4.1.html

[27] M. K. Geldenhuys, L. Thamsen, K. K. Gontarska, F. Lorenz, and O. Kao, "Effectively testing system configurations of critical iot analytics pipelines," in *2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, December 9-12, 2019*, C. Baru, J. Huan, L. Khan, X. Hu, R. Ak, Y. Tian, R. S. Barga, C. Zaniolo, K. Lee, and Y. F. Ye, Eds. IEEE, 2019, pp. 4157–4162. [Online]. Available: https://doi.org/10.1109/BigData47090.2019.9005504

[28] M. K. Geldenhuys, L. Thamsen, and O. Kao, "Chiron: Optimizing fault tolerance in qos-aware distributed stream processing jobs," in *IEEE International Conference on Big Data, Big Data 2020, Atlanta, GA, USA, December 10-13, 2020*. IEEE, 2020, pp. 434–440. [Online]. Available: https://doi.org/10.1109/BigData50022.2020.9378474

[29] F. Lorenz, M. Geldenhuys, H. Sommer, F. Jakobs, C. Lüring, V. Skwarek, I. Behnke, and L. Thamsen, "A scalable and dependable data analytics platform for water infrastructure monitoring," in *IEEE International Conference on Big Data, Big Data 2020, Atlanta, GA, USA, December 10-13, 2020*. IEEE, 2020, pp. 3488–3493. [Online]. Available: https://doi.org/10.1109/BigData50022.2020.9378138

[30] M. Haug, F. Lorenz, and L. Thamsen, "Gral: Localization of floating wireless sensors in pipe networks," in *To appear in the Proceedings of the 9th IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2021.