Databases in Edge and Fog Environments : A Survey

LUÍS FERREIRA, INESCTEC & U. Minho, Portugal
FÁBIO COELHO, INESCTEC & U. Minho, Portugal
JOSÉ PEREIRA, INESCTEC & U. Minho, Portugal

While a significant number of databases are deployed in cloud environments, pushing part or all data storage and querying planes closer to their sources (i.e., to the edge) can provide advantages in latency, connectivity, privacy, energy and scalability. This article dissects the advantages provided by databases in edge and fog environments, by surveying application domains and discussing the key drivers for pushing database systems to the edge. At the same time, it also identifies the main challenges faced by developers in this new environment, and analysis the mechanisms employed to deal with them. By providing an overview of the current state of edge and fog databases, this survey provides valuable insights into future research directions.

CCS Concepts: • **Information systems** → **Sensor networks**; **Database management system engines**; **Database design and models**; *Storage replication*; • **Security and privacy**;

Additional Key Words and Phrases: Edge Computing, Fog Computing

ACM Reference Format:
Luís Ferreira, Fábio Coelho, and José Pereira. 2024. Databases in Edge and Fog Environments : A Survey. *ACM Comput. Surv.* xx, x, Article xxx (May 2024), 40 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Database Management Systems (DBMSs) have become the *de facto* solution for structuring, storing and querying data, allowing developers to focus on application logic, and thus forgo concerns of, e.g., data integrity and consistency. Many of these systems are now being deployed in cloud datacenters, with the proportion of corporate data offloaded to such services having increased from 25% in 2015, to 60% in 2022 [106]. The cloud offers ready to use database solutions, as well as a myriad of infrastructure and service options, which allow database administrators to delegate part or all of the database management concerns over to the cloud provider. Moreover, resources may be allocated automatically, ensuring optimal resource usage. Even so, the cloud is not without its disadvantages. Public cloud datacenters are only available at a few geographical locations, leading to significant latency penalties [19]. Furthermore, network limitations on the user side, such as limited bandwidth, or intermittent connectivity, make cloud offloading harder [68, 113]. Finally,

Acknowledgements: This project has received funding from the European Union's Horizon Europe research and innovation programme under the Grant Agreement 101136216. It is also funded by National Funds through the FCT — Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) PhD grant identified by doi https://doi.org/10.54499/PD/BD/151402/2021

Authors' addresses: Luís Ferreira, INESCTEC & U. Minho, Braga, Braga, Portugal, luis.m.ferreira@inesctec.pt; Fábio Coelho, INESCTEC & U. Minho, Braga, Braga, Portugal, fabio.a.coelho@inesctec.pt; José Pereira, INESCTEC & U. Minho, Braga, Braga, Portugal, jop@di.uminho.pt.

0360-0300/2024/05-ARTxxx $15.00
https://doi.org/XXXXXXX.XXXXXXX

public cloud environments are managed by 3rd parties, and span across international borders, which can pose a privacy concern [10, 105]. Private cloud datacenters are a possible alternative, but not all companies possess the scale to justify that approach [128].

As a substitute or complement to the cloud, edge and fog computing systems place nodes with computing and storage capabilities closer to the end-users, taking advantage of their proximity to overcome some of the cloud's and other centralized systems' limitations [126]. However, they do so at the cost of greater implementation complexity. Moreover, the areas of edge and fog computing are home to a significant amount of conflicting definitions. There is no clear definition for the boundaries of an edge environment, or whether the terms *edge* and *fog* are even distinct (e.g., [35] vs [102] vs [126]). As such, understanding the role of databases in such a volatile environment, and building a classification that can help developers understand the different research axis of the field is a complex challenge. Classifying systems requires understanding the history of edge and fog computing, and establishing a definition upon which further classification can be built.

1.1 Background in Edge and Fog Computing

Edge and fog computing present an alternative to systems that offload tasks exclusively to a central location, e.g., cloud provider, by adding computation and storage nodes closer to the users. This provides multiple advantages such as lower latency and possibly increased privacy. Although the terms edge and fog often establish a parallel with the cloud, the expression *edge* actually predates the first cloud service, meaning the location closer to the end devices where tasks are executed.

In 1991, Mark Weiser presented a seminal paper introducing the concept of ubiquitous computing [119], describing a world where computers are present in most objects we interact with, working transparently to improve our lives. This would form the basis for edge and fog computing, and adjacent concepts such as the Internet of Things. Between 1997 and 2001, Noble, Satyanarayanan and colleagues followed on the footsteps of Mark Weiser, renaming the concept to pervasive computing. They proposed that, to achieve pervasive computing, one would need to offload tasks from mobile devices to surrogate servers, deployed in common venues such as coffee shops or airport lounges [78, 95]. After public cloud datacenters made their debut in 2006 [74], Satyanarayanan established a parallel with the new paradigm, proposing the use of cloudlets [97], "a cloud in a box" deployed close to mobile users. Cloudlets would avoid WAN delays, jitter, congestion, and failures which can occur when offloading tasks to the cloud, as well as other issues such as limited bandwidth. Concurrently, in 1999, Akamai launched the first Content Delivery Network (CDN) [3], which in contrast to a centralized web server deployed web servers in various geographical locations to achieve lower response latency, having likely been the first to use the term *edge* in an interview in 1995 [18]. Both pervasive/ubiquitous computing's and CDNs' research efforts chose the edge as a substitute for the cloud to solve latency, bandwidth and other cloud limitations, culminating in the rise of edge computing.

In 2012, Cisco introduced the complementary yet distinct paradigm of *fog computing* [14], where public clouds were extended to the edge by providing compute, storage or networking services. Although they share similarities, the *fog computing* standard that was later published in 2018 classifies *edge* and *fog computing* as distinct concepts [23]. From there, multiple terms have emerged, such as mist computing [26], which moves computation to the end devices themselves; or multi-access edge computing, with its own set of standards [37], which places computation and storage capable devices within Radio Access Networks (RAN), e.g, 5G towers. The welter of concepts originates conflicts amongst publications, with definitions varying considerably. To avoid confusion we consider only the concepts of *edge* and *fog computing* as defined in Section 1.2.

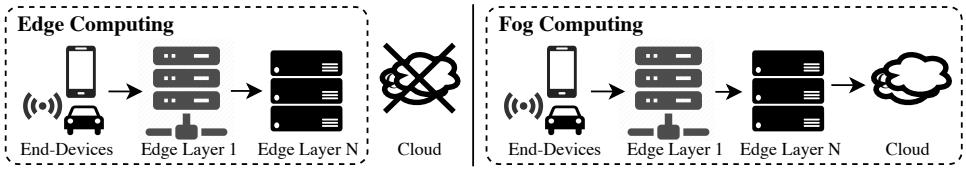

Fig. 1. Edge as replacement of cloud computing. Fog as extension of cloud computing.

1.2 Defining Edge and Fog Computing

Edge and fog computing place resources closer to end devices, overcoming the limitations imposed by significant geographical and network distances. In both approaches, developers must decide how to best distribute tasks amongst the different available layers. Depending on use case characteristics, users must choose, for each layer, whether to keep data temporarily or permanently, with what granularity (e.g., raw, filtered, or aggregated), the level of consistency to ensure between layers, and where to perform processing tasks such as filtering and query optimization.

The distinction between *edge* and *fog computing* varies from publication to publication, but three definitions stand out in related work. The simplest definition assumes the terms are equivalent, and thus can be used interchangeably [90, 102], translating to computation that occurs outside of a centralized location. A second approach considers edge, fog and cloud as hierarchical layers that can co-exist in a single system, often differentiated by where they sit in the network [35, 75]. Distance to the end devices increases as we go up to the cloud, as do the amount of resources available. The inverse occurs when going down to the edge, with fog serving as an intermediate layer of medium distance and medium-sized resources. Finally, a third approach, and the one we adopt for this work (as depicted in Figure 1), considers *edge* and *fog computing* to be disjoint concepts. In the *edge computing* paradigm, systems move all tasks to edge nodes, effectively replacing the centralized resources with the edge. On the other hand, the fog computing paradigm combines centralized and edge nodes (e.g., edge + cloud) to extract advantages from both processing planes. This work adopts this definition, as it is the one provided by the IEEE published standard for fog computing, where it reads: *Fog computing also is often erroneously called edge computing, but there are key differences. Fog works with the cloud, whereas edge is defined by the exclusion of cloud* [23]. This standard is authored by the OpenFog Consortium [39], which is composed of, amongst others: Cisco Systems, Intel, Microsoft, Princeton University, Dell, and ARM Holdings. This definition is also adopted by other relevant publications in the area [50, 126].

Having settled on a definition for edge and fog computing, the question remains for what are the boundaries of a system's edge environment. In this work, it is considered that the edge is formed by the nodes that tasks are delegated to in alternative to a centralized location. The edge environment starts at the end-devices that produce or consume data, while its limit depends on the specific application domain. For example, in the case of Autonomous Vehicles, vehicles and roadside units one network hop from the vehicles compose the edge nodes [49]. For Points of Presence, the edge extends to small datacenters that can be located several network hops away from end-devices [67].

1.3 Motivation

Edge and *fog computing* have renewed their popularity in the past few years [41, 96], emerging as an alternative or complement to the cloud, that can help tackle evolving application demands. Hardware developments allowing more complex tasks to be undertaken by small factor, low-powered devices that can easily be deployed at the edge play an important role in the growing interest. In these new environments, database systems have received little attention, with the

storage plane being neglected in favor of the processing plane. Most often the edge is used to produce, filter, aggregate, and compress data, [50, 93, 126] and to produce automated responses according to most recently read values. Storage and more complex data analysis is delegated to the cloud due to its vast amount of resources, and ready to use software solutions. However, the edge provides data storage advantages which may be essential to applications as their needs grow:

Connectivity. In an edge environment, multiple connectivity limitations can be in effect, making a *cloud-only* database deployment not viable. An **intermittent connection** means data may have to be stored at the edge, at least temporarily, while the connection is reestablished. To avoid data loss during this period, data should be temporarily persisted and possibly replicated. The edge database can also ensure service availability while the connection is interrupted. **Low bandwidth** may also be a limitation. A long range, low power wireless network, such as LoRa [94], may limit the bandwidth at which a node or cluster of nodes is able to communicate with external nodes. In this case, a database may be used to persist local data, and efficiently pre-process it so that a more compact form can be synced with a central node for analysis. Finally, the system may operate in an **isolated environment**, without the possibility to contact a cloud datacenter, thus requiring that the database be deployed entirely at the edge.

Privacy. Data protection acts such as PIPA [57], DSG [79], LGPD [80], POPIA [81] and GDPR [112] are a growing requirement for data privacy. Often, the term privacy refers to authentication or encryption, i.e., security concepts. From a security perspective edge nodes are more susceptible to physical attacks, and constrained resources equate to limitations in supported security algorithms [117]. However, the edge also provides an important advantage in data placement. The ability to deploy databases in fog/edge nodes massively increases the options for geographical location, and infrastructure ownership. This provides greater flexibility in scenarios where data is under strict regulations, or where user privacy preferences are of concern [105].

Latency. Lower latency may lead to a better user experience, or provide an important advantage over competitors, thus justifying placing databases at the edge [6, 15, 107]. However, quantifying the minimum latency gain that constitutes an advantage is non-trivial. The difference between edge-enabled and cloud-only latencies may be very small, making the decision to place the database at one location or the other very difficult [90].

Energy. In scenarios where edge nodes are battery powered, and communication is one of the most energy consuming stages [111], diminishing the number of messages exchanged equates to a longer lifetime of the edge nodes' battery. Keeping data stored in the nodes that generate it, and only retrieving it on a per-query basis, can lead to a significantly longer lifetime for the sensors.

Scalability. Data is being generated at an unprecedented scale, and IoT applications are expected to generate up to 80% of all data by 2025 [92]. Uploading it all to the cloud would cause major network bottlenecks [4]. Storing data at the edge; applying deduplication, filtration and aggregation; and only uploading data on an as-needed basis mitigates this bottleneck and contributes to reduce the amount of computation to be done at cloud servers, allowing systems to scale further.

1.4 Contributions and Scope

Existing surveys targeting storage at the edge focus on particular database use cases (e.g., Sensor Networks [29]), or look at storage systems for the edge in general [76], not focusing strictly on database systems. This survey fills the gap that exists between them, providing an overview of databases targeting edge and fog computing, and helping guide future research in the area. The

appropriate edge database is selected by identifying its **application domain**, and understanding its **latency**, **hardware**, **energy** and **privacy requirements**. Thus, we pose the following questions:

Q1 - What are the application domains for databases at the edge? Database development under edge environments is highly dependent on the application domain. However, the identification of those domains, and their main characteristics, i.e., what contributions are made to make systems suitable for a given use case, is absent from the literature. This makes identifying keywords and categorizing new systems more difficult, which in turn can slow down research.

Q2 - What are the hardware constraints databases face on edge nodes? Edge implementations are highly dependent on node hardware characteristics. While edge nodes generally have constrained and asymmetric resources (i.e., varying between nodes) [16, 124], no analysis has been done on the concrete hardware capabilities of nodes in each application domain.

Q3 - How big is the latency advantage provided by the edge? Latency is recurrently used as the motivation for extending database systems to the edge [5, 122], but there is a lack of analysis on the **advantage in latency provided by the edge** in various application domains. There is a need for clearly understanding: the **minimum latency gain** needed to provide an application with a meaningful advantage; and the trade-off between the complexity of augmenting an established system with edge resources, and the (often low) latency advantage provided by the edge.

Q4 - How can edge systems help improve privacy? Privacy is an increasingly influential aspect of data storage, as regulations and increased user privacy awareness lead to restrictions in data placement. Extending database systems to the edge offers the possibility for data to be placed closer to the user, as well as for it to be stored under varying degrees of detail across layers. The survey concludes that this aspect is not considered in most systems, and that it could constitute an important research axis in future work.

Q5 - How do energy limitations affect edge databases? Edge nodes often operate with limited energy, however exact limitations are not well-defined. For battery powered nodes, the objective is clear, i.e., to prolong the lifetime of the device, however most edge database nodes are not battery powered. So, despite some publications exploring the dimension of energy consumption and its impact, energy requirements are not obvious when mains-powered edge devices are often employed. This survey points out several shortcomings in the analysis of energy consumption.

Taking into account the identified questions, this survey presents an application domain analysis of edge and fog oriented DBMSs. It categorizes systems into application domains, and provides a comparison at the hardware, latency, energy and privacy level. Moreover, it discusses how these characteristics affect database implementation, and deployment patterns. Finally, it overviews the evaluations of these systems to analyze what tools are available to researchers, and at what scale evaluations should be conducted. Specifically, this survey provides the following contributions:

- Identifies and divides edge and fog database solutions into application domains, namely: Sensor Networks, Points of Presence, Autonomous Vehicles and Other.
- Analyzes the main architectures used in these database solutions, both from a deployment and from an internal component perspective. Four common deployment patterns are established: edge, multi-layer, isolated and local quorum.
- Analyzes latency values from the evaluations of surveyed publications to understand the latency advantage associated to storing/processing data at the edge when compared to the cloud. Furthermore, it identifies relevant latency requirements, i.e., establishes what latency improvements constitute significant advantages to different application domains.
- Compares hardware resources considered in surveyed publications, to understand the hardware limitations and asymmetries databases face in the edge.
- Identifies the (lack of) privacy mechanisms considered in each publication.

- Determines the energy limitations of edge database nodes, and evaluates the optimizations employed to deal with them.
- Analyzes the evaluations conducted in surveyed publications, from a perspective of simulation/emulation tools used, workload considered, and scale.

The focus is on solutions employing at least one edge storage node. Systems which use the edge for DBMS data pre-processing, but not for storage, are not considered. Also, we only consider systems that specifically target deployment in edge or fog environments. Solutions targeting cloud environments, i.e., InfluxDB [52], are not considered for analysis. Finally, this survey does not target storage systems with different classifications, such as file-systems or object storage systems.

The remaining sections are structured as follows. Section 2 explores edge-enabled database systems, separating them into application domains. Section 3 quantifies latency performance and hardware constraints in each application domain, as well as evaluates systems' energy limitations and privacy concerns. Section 4 explores database internal components, and which components are targeted by each application domain. Section 5 analyzes the scale, workloads, and simulation and emulation tools being used in the evaluation of edge database publications. Section 6 contextualizes this work amongst already existing surveys. Summarizing the work presented, Section 7 presents key take-aways and future research directions.

2 APPLICATION DOMAINS

The edge is composed by a plethora of application domains,[1] whose needs and particularities steer the development of new database systems. This section identifies those domains, and explores corresponding publications, highlighting which mechanisms make them appropriate for their given domain. In total, three categories have been identified: Sensor Networks, Points of Presence, and Autonomous Vehicles. Solutions which do not fit into any of these domains are classified as *Other*.

2.1 Sensor Networks

Sensor Networks comprise the main research area for novel databases developed specifically for the edge. This is to be expected, as the continuous growth in the number of endpoints or Internet of Things (IoT) devices with embedded computing capabilities is expected to duplicate by 2025, and represent close to 80% of all data [92]. These systems can be separated into two types: Time-Series Database Sensor Networks, and Search-Engine Sensor Networks.

2.1.1 Time-Series Database Sensor Networks. Sensor networks are used to monitor environments, and leverage the collected data for: future event prediction, pattern analysis, automatic alarm generation, event triggering, and real-time metric visualization [13, 88, 118]. All of these objectives require the notion of time, or of order in time of the acquired metrics. So, monitoring data is accompanied by a corresponding timestamp which specifies when a given measure occurred. Furthermore, for large deployments, billions of data points can be generated in a single day [47], so there is a need for efficient storage and management mechanisms. Time-Series Databases (TSDBs) are specially suited to deal with time-stamped data, offering high throughput ingestion for a big number of parallel input streams, faster processing of time range operations, such as aggregate value functions based on time-intervals, and optimized compression for this class of data.

Most systems offload all data directly to a TSDB in the cloud [17, 89]. Applications which require the management of this type of data closer to end devices are special cases which deal with limited

[1]The following abbreviations are assumed to be interchangeable with the identified application domains: **SN-SN** - Sensor Network Sink Node; **SN-IN** - Sensor Network In-Network; **SN-P** - Sensor Network Predictive; **SN-I** - Sensor Network Isolated; **SN-SE** - Sensor Network Search-Engine; **PoP** - Points of Presence; **AV** - Autonomous Vehicles. The (P) qualifier is used to denote a practical scenario, where a database is deployed at the edge as part of a bigger system.

connectivity problems, such as: intermittent communication or even absence of WAN connection. They can also exhibit tight bounds on response latency, or generate high amounts of data which could exceed network bandwidth capabilities.

Example use cases. In Smart Healthcare, wearable devices are used to monitor patients vital signs, with edge gateways hosting local databases for data originating from such devices [90, 122]. The derived advantages are: postponed replication to cloud main storage, and high-availability, upon WAN connection interruption; increased data privacy; lower load on cloud's resources and network bandwidth due to data aggregation/filtering/anonymization at the edge level (e.g., in the hospital itself); and lower latency for automatic response generation (e.g., alarm generation).

In Industrial IoT (IIoT), databases at the edge are used to manage shared state between machinery equipment, e.g., CNC machines [13], in order to optimize their manufacturing process. By hosting the database at the edge, lower latency can be achieved.

These systems show how databases can serve various IoT applications, but they constitute simple proofs-of-concept, and conduct evaluations targeting small scale deployments. Furthermore, they opt for the use of well-established, and cloud-oriented, DBMSs, which are not the best option for resource constrained nodes. On the other hand, novel solutions specifically target these concerns. **Hardware** and **energy** contraints are the two focus points of novel TSDBs for Sensor Networks, and four common architectures can be discerned:

Sink Node Architecture (SN-SN) In the Sink Node architecture, sensors offload all data directly to an edge node called the sink node. The system can have multiple sink nodes, each responsible for a subset of sensors. This node either pre-processes data, and then uploads it to a cloud TSDB, for permanent storage and further analysis; or persists the data locally, with a combination of the two also being possible. The sink node lowers the load on network bandwidth and cloud's resources, by pre-processing data before upload. It can also serve as a temporary persistent store of data, in case of WAN connection loss, as well as serve client requests locally.

SN-SN Database Systems The main difference between SN-SN and cloud TSDBs is the use of the sink node. In order to provide higher ingestion throughput, and query performance, solutions maximize I/O operation size, provide specialized indexes and highly compress data. Firstly, we present EdgeDB [124] and VergeDB[85], which focus strictly in providing edge-side processing and storage of data, thus being classified as edge computing systems.

EdgeDB [124] merges correlated sensor data streams (e.g., data from the same location, or from same sensor type) into tablets, and then organizes tablets with the same time-interval into tablet groups. These tablet groups are then stored in a TMTree, a tree structure ordered by the time-interval of the tablets. Joining correlated data and ordering it chronologically improves query latency, as it favors big sequential read I/O operations. Furthermore, by writing multiple tablets at a time, I/O operation size is increased, maximizing the FLASH storage's throughput. A specialized index, TPEI, indexes tablets based on their time-intervals and pre-calculates aggregated values (e.g., sums), allowing for faster time-interval and aggregation queries. TPEI indexes follow a hierarchical subtree structure of daily, weekly, monthly and global indexes. To limit memory usage, only the most recent TPEI trees are kept in-memory (e.g., the daily tree for the last 7 days), with the remaining ones being written to disk.

Working as a middleware, **VergeDB** [85] offers pre-processing at the sink node level, implementing multiple compression algorithms. The algorithms used allow for analytical operations to be conducted without having to decompress the data. The same data may be compressed using different algorithms, namely through precision-aware, subsampling and similarity-preserving

compression. Filtering, data mining and anomaly detection are some of the operations that can be performed without decompression. At times of higher demand, data which does not fit in-memory is temporarily kept in a local RocksDB datastore [34].

Using both cloud and edge devices to process data, ApacheIoTDB [115], Respawn [16], ModelarDB [54] and TorqueDB [40] all fall into the category of fog computing systems.

ApacheIoTDB [115] maximizes I/O write throughput by first writing the most recent values unordered in an in-memory data structure, with a WAL log ensuring persistence. Data is then ordered and written in bulk to FLASH storage, using vectorization to further improve throughput. Data is provided in TsFiles, ApacheIoTDB' own format, pre-calculating statistics such as averages, counts and others to improve query performance. The use of PISA and KV-Match indexes aid in aggregation, down-sampling, and sub-sequence similarity matching. After being written into the TsFile format, data can be offloaded to a cloud node for more complex analysis.

Respawn [16] targets the specific case where edge node bandwidth is very limited, and requests to edge nodes incur higher latency than to cloud nodes. Data is stored in multiple resolutions, starting with the raw data points, and downsampling (i.e., aggregating) the values at each level by a power of 2. This allows time to be saved on queries targeting aggregated data. The challenge in such an environment is deciding which data to cache at the cloud. The decision is based on the data that is most likely to be accessed, which for the case of Respawn are the levels with greater data aggregation. Upload is done according to available bandwidth and edge node load. When a client request is issued, a dispatcher uses bloom filters [11] and other resources to decide whether to route a request to the cloud or to a particular edge node.

ModelarDB [54] focuses on aggressive data compression at the edge, due to constrained network bandwidth, while ensuring minimal data loss. The system integrates multiple data stores and query engines, allowing any combination of components or storage adaptors. Edge nodes use the H2 embedded database as the query engine, and either a JDBC-compliant RDBMS, Apache Parquet or Apache ORC local files. For cloud, Apache Spark acts as query engine, and Apache Cassandra, Apache Parquet or Apache ORC files are written to HDFS. For efficient representation of data, records at the edge are transformed into representative models with bounded error margins using the Group Online Lossy and lossless Extensible Multi-Model (GOLEMM) compression method. The algorithm splits the data into sub-segments of contiguous time intervals. The split is done so that each sub-segment is as big as possible while still being able to be represented by one of the available models within the maximum allowed error. Data originating from different sensors which shares similar values can share the same model, further increasing the level of compression.

TorqueDB [40] aims to provide similar performance to that of a cloud TSDB, by efficiently distributing the load amongst many edge devices. The architecture makes use of two existing systems: InfluxDB [52] and ElfStore [75], implemented over a series of node clusters. For a given query, a coordinator node generates a series of sub-queries which are independently executed at each of the involved node clusters. In each one, ElfStore loads pre-identified data blocks from its storage to InfluxDB, where the sub-queries are executed. The results are sent back to the coordinator, who either appends or aggregates them, relaying the final result to the user.

The sink node architecture does not consider the sensors as part of the database system, with data being processed and stored on above layers, delegating to sensors the single task of transmitting readings over to the sink node. As sensors are battery-powered, energy expenditure is of major concern, so including them in database design should be taken into account.

In-Network Processing Architecture (SN-IN) As an alternative, the in-network processing architecture focuses on extending the database system to the sensors themselves, aggregating,

filtering and even storing data at the sensor level. This lowers exchanged messages with upper layers, reducing the energy consumed by sensors and lowering the load on upstream nodes.

SN-IN database systems Reviewed publications do not offload data to the cloud, placing all reviewed systems in the edge computing category.

Cougar [27, 125] was the first system to advocate in favor of this model. When a client issues a query, an execution plan is generated at an edge node with more capacity than the sensors. The plan is then executed using the pre-selected sensors, splitting the Query Engine's planner and plan executor (Figure 4) between sensors and edge nodes. Furthermore, the concerns of the query planner change, as contrary to customary systems, the plan is made to consider energy consumption and nodes' resources. Cougar also explores the idea of sharing intermediate results across similar queries, and coordinating sensor activation schedules to avoid transmission collisions.

TinyDB [69] follows a similar split-engine and sleep-schedule optimized approach, but provides the advantage of user-system interaction through an SQL-like query language. It also adds new insights into in-network querying. *LIFETIME* queries adjust the sampling and transmission rates of sensors to ensure battery lifetime exceeds or at least matches the time of the query. For example, the query plan may determine that in order to run a given query over a period of 24 weeks, sensors cannot exceed a maximum sampling rate of 10 samples/day. This type of queries are born from the insight that communication and non-sleep time are the main consumers of energy, and as such the sample rate, which dictates how often a sensor must be activated and relay messages, is a fundamental mechanism for energy control. Furthermore, Semantic Routing Trees (SRTs) determine optimal data dissemination paths for queries involving pre-determined static attributes.

Antelope [111] leverages the evolution in CPU capabilities and flash-memory storage capacity to deploy each sensor as an independent relational database, capable of insert, remove, select, join and schema operations. The main rationale is that data transmission and active sensor periods incur the biggest energy consumption, so by storing all data locally and only querying for the specific data that is required, energy consumption can be minimized. Targeting resource constrained sensors, Antelope's base logic uses only 4KB of memory, and its entire codebase occupies only 21KB. A virtual machine, LogicalVM, deploys the database's propositional logic in efficient bytecode format to accelerate query execution. Performance is further improved by implementing a kernel with cooperative scheduling, allowing multiple queries to be handled concurrently, and by employing a series of specialized indexes. It considers a MaxHeap index for its low energy consumption and wear on flash storage; an Inline index efficient at dealing with time-series and ordered data; and an hash index that stores frequently accessed relations in-memory, for faster access.

Future solutions should explore the combination of sink node and in-network processing models to render a database system optimized for both the sink node and sensor layers.

Model Prediction Architecture (SN-P) The model prediction architecture is an innovative multi-layer approach to sensor network querying that uses Machine Learning models to avoid contacting sensors for a large amount of queries.

SN-P database systems Making use of proxy nodes with rich communication, computational and storage resources, **Presto** [63] avoids data communication with sensors by serving requests based on predictive time-series models. The proxy node is used to answer queries when models are able to provide predictions with a given certainty level. Otherwise, it will contact the concerned sensors, pulling the value from them. The sensors themselves also keep a copy of the prediction model, and periodically check whether the predicted values are within an acceptable margin of

error from the actual sensor readings. If not, the measured value is pushed to the edge node, so that anomalous values are still made available at the proxies. Pushed, pulled and predicted values are cached indefinitely at the proxy, to avoid retrieving or predicting the same value twice. Once a predifined number of pushed or pulled sensor values are received at a proxy, its model is retrained and the new parameters are relayed to the sensor, following an online learning strategy. As all operations are conducted at the edge, this system is classified as an edge computing application.

Future research on Sensor Networks should take into account hardware [64] and Machine Learning advancements [118], to expand predictive based approaches to more complex data patterns. It could also consider **sharing models for sensors with similar values**.

Isolated Architecture (SN-I) Under certain scenarios, e.g., scientific field experiments, a sensor network may operate without a permanent connection to a storage sink node or cloud provider, and be forced to follow the isolated architecture.

SN-I database systems Facing limited storage resources, nodes may not be able to store all the data they generate, and so, alternative methods to avoid data loss must be explored. Since systems focus on the edge component of data handling, they are classified as edge computing applications.

Vasilescu et al. [113] presents a unique underwater environment where nodes operate isolated from a network connection. Underwater sensors cannot communicate through radio communication nor with each other or gateways, as sea-water heavily attenuates radio signals. To relay data for analysis to a centralized location, Autonomous Underwater Vehicles (AUV) are used as data mules. In the context of SN-I systems, data mules are devices which in proximity or by physical connection to a sensor are able to retrieve data, and immediately or *post factum* transmit it to a storage location. The AUVs make use of acoustic signals for: broadcasting events from sensors to AUVs; low-bandwidth communication; and 3D localization of sensors. As acoustic communication does not enable enough bandwidth for data offloading, optical communication is also employed, providing higher-bandwidth for short-range and line of sight communication.

Envirostore [68] focuses on sensor deployments without an established network connection, e.g., scientific field experiments employing temporary sensor deployments. Sensors may have different resources, and some may generate more data than others, so Envirostore proposes data redistribution amongst sensors, by locally evaluating whether there is a need to offload data to neighbors. A trade-off is played out between the energy cost of data transmission, and remaining storage space, as both are required to operate. When offloading data, sensors limit the size of transmitted data, and partially randomize the selection of the receiving sensor, to avoid overloading the node with most available space. Doing so could lead to data being returned to the original sensor, creating a ping-pong effect which would incur unnecessary messages. Envirostore also considers data mules which, when available, allow sensors to offload their data to them, highlighting the importance of exploiting moving entities already present in the system, e.g., using wild animals which live in the zone being sensed.

2.1.2 Sensor Network Search-Engines (SN-SE). Sensor networks can be used to store and retrieve data by providing search-engine functionality. Sensors in the network hold information, usually associated to an object they are attached to. Search-engines are then tasked with seeking the best match to a given textual query, which a user can issue to points of access in the network.

SN-SE database systems Search-engines allow multiple types of filtering criteria, namely: keywords, values and spatio-temporal cues. They can also adopt either push or pull-based updates, exploring the trade-off between number of queries posed to the system and number of updates

generated by its members. As surveyed publications in this application domain rely only on edge nodes, they belong to the edge computing category.

Snoogle [117] allows users to find the location of a set of sensors that most resemble a given search term. The architecture follows a three layer approach: Sensors, Index Points (IPs) and Key Index Points (KeyIPs). Sensors are attached to objects and store a textual description of the object or its contents, as well as a frequency distribution of the words used in its description (metadata). IPs identify a given location, e.g., a drawer, indexing search terms from sensors' metadata to sensors present in the location they monitor, while KeyIPs aggregate data from multiple IPs into a single node, enabling distributed queries over multiple IPs (i.e., multiple locations). IPs and KeyIPs index data using bloom-filters, following a push-based update strategy that uses timeout counters and periodic beacons from sensors. This can lead to a reduced lifespan of the battery powered sensor network, but promotes improved freshness in the indexed data. KeyIPs identify and correct inconsistencies, in the event that two IPs report the same sensor. To improve I/O performance, IPs and KeyIPs employ in-memory buffers which only flush to disk (i.e., to bloom-filters) when full. Bloom-filters maintain a map of search terms to bucket addresses in-memory, while the buckets that store references to the sensors are kept in FLASH storage.

In contrast to Snoogle, **Dyser** [83] proposes a pull-based approach, where sensor information is only retrieved when queried. The strategy proves better for systems where the number of sensor updates outweighs the number of client queries, as the smaller number of exchanged messages improves the sensors' battery lifetime. Dyser holds three components: a sensor-layer, a gateway layer and a search-engine. To minimize the number of contacted sensors, gateways periodically build prediction models advising on which sensors are more prone to have the requested state, based on a fixed size time-window of sensor states. The search-engine then periodically crawls the gateways to get their probability models. When searching, users reach the search-engine which contacts gateways and corresponding sensors in descending order of probability of matching with the requested state. When the number of requested matches is found, results are returned to the user. Contrary to Snoogle, searches are done by sensor values, instead of through generic text statements. For example, to query the system for places with low occupation and noise, the corresponding query would translate to: *"people:few loudness:quiet"*.

Search-NDNoT [98] combines push and pull-based approaches to provide lower latency in continuously monitored values, while saving energy on sporadically queried data. The architecture considers two mediator layers and a central server. In the push-based approach, the first mediator periodically reads objects' RFID tags and aggregates data at a specific location. The second layer aggregates data for a group of locations. Finally, the central server keeps a database of retrieved data. In pull-based sensors, queries use Named Data Networking (NDN) [1], an alternative network architecture to TCP/IP, to retrieve sensors' state. Using NDN for pull-based queries mitigates IP address exhaustion induced by exponential growth of *Things* (in the IoT context), as the unique name-based identification of nodes provide a much bigger identifier pool. The central server serves as host for all user queries.

Each of the previous systems focuses on a specific query type. Snoogle and Search-NDNot support the use of keywords to retrieve objects' locations, matching search terms to textual descriptions held in sensors. Dyser, supports value-based searches, where sensors are retrieved based on desired state. The **IoT-SVK Search-Engine** [31, 32] goes one step further, proposing the use of parallel indexes to support multi-modal retrieval conditions, namely: spatial-temporal, value-based or keyword-based. The system is composed of sensor, storage and indexing layers. The sensor layer includes both numeric sensors, reporting values such as temperature and humidity, and multimedia sensors, e.g., audio and video devices. Data vitalization is further performed over multimedia content, generating complimentary data, e.g., signaling the presence of a given object in an image. At the storage

layer, IoT-SVK uses multiple Raw-Data Storage (RD-Store) to store data as sampling values. Each sample contains information on time of data retrieval, location, the sensor's schema (i.e., name and type of value fields), collected values and a pointer to a raw file (multimedia content). Each sensor is assigned to an RD-Store and represented through an ObjectDataRecord, composed of an ID, description and set of sampling values. At the index layer, three index structures enable the three query types. The FTKB+-Tree uses information from object (i.e., sensor) IDs and descriptions to form a distributed index that enables keyword based queries. The ISTR-Tree index uses a master index to distribute a geographical area between a series of nodes, in which $S^3T - Trees$ index the positions and timestamps of samples to static and dynamic position subtrees, enabling spatio-temporal queries. Finally, the $SCVSKB^+ - Tree$ indexes raw sample values to enable value-based searches. Users query the master index, which in turn parses and evaluates which indexes to invoke.

2.2 Points-of-Presence

The term edge was first used in the context of Content Delivery Networks [18], when the creators of the first CDN, Akamai [30], proposed distributing web servers across multiple edge locations to improve latency and throughput. CDNs are composed of servers located at various geographically dispersed locations, commonly referred to as Points-of-Presence (PoPs). Aside from Akamai, Amazon [8], Google [22] and Microsoft [9] also offer this service. The use cases for databases in Points of Presence are similar to those of geo-replicated cloud databases, but by placing nodes at the edge, new advantages can be reaped. At PoPs, databases concern themselves with maximizing the latency advantage provided, which can be achieved by understanding the new dynamics found at edge environments. For example, predicting user movement patterns can help ensure data can always be found at the nodes nearest to them. These systems are often composed of cloud oriented DBMS instances running specialized replication protocols.

PoP database systems PoPs are composed by servers which, although less capable than those of cloud datacenters, still have considerable resource capacity (see Table 1), making them suitable to run cloud-oriented DBMSs. Most such systems constitute edge computing scenarios, since cloud datacenters are replaced by smaller infrastructures closer to the users. This is the case for SEQ/Hybrid [67] and Pfandzelter et al. [87]. However, certain systems still make use of central nodes deployed in cloud providers to act as coordinators. As such, some PoP based systems are instead based on the fog computing approach, namely DBProxy [5] and Meiklejohn et al. [72].

SEQ and Hybrid [67] provide transactional semantics, and strong consistency guarantees, while maintaining the low latency advantage of the edge. In both algorithms, read transactions can be commited using only the contacted replica. Writes, on the other hand, use a distributed variant of Snapshot Isolation, i.e., 1-copy-SI, which requires a single WAN exchange between replicas. In the case of **SEQ**, on commit, the writeset of the transaction is sent to the sequencer, a centralized service running on one of the replicas. The sequencer checks for conflicts, and either aborts the transaction and informs the originating replica of the decision, or it validates the transaction and transmits the writeset to all replicas in a FIFO fashion (including to itself). The downside of this approach is that if the sequencer crashes, progress is halted and some transactions not propagated to other replicas may be lost. **Hybrid** avoids this single point of failure by using uniform reliable multicast, which guarantees that all replicas in a messaging group receive all messages in the same order, unless they crash. Replicas are split into a main cluster, and a series of edge clusters. When a transaction is submitted to the main cluster, it is checked for conflicts and, in the case of success, is replicated to all replicas in the main cluster using uniform reliable multicast, as well as propagates those changes to the sequencers of the edge clusters. At edge clusters, a sequencer identical to the one used in SEQ maintains the sequence of operations for replicas to apply in

the local cluster. When a write transaction is submitted to a replica in the edge cluster, the local sequencer is used to check for conflicts within local transactions, and in case of success, is sent for further conflict verification in the main cluster. The advantage is that the main cluster won't have to check for conflicting transactions already tested by the local sequencer, taking some load off of the main cluster. Once the transaction is approved at the main cluster (or aborted), the writeset of the transaction (or abort decision) is propagated at the main cluster as well as to edge clusters.

While in the cloud all requests are redirected to a central point, at the edge, requests are routed to the closest node, which may vary for non-static users. **Pfandzelter et al.** [87] proposes preemptively replicating data to and between edge nodes according to the traveling patterns of mobile users, maximizing the hit rate of requests posed to the edge nodes closest to them. In the proposed architecture, a middleware running on client nodes would collect metadata information about accessed data and location of accesses, relay that data to edge nodes, and then edge nodes would use machine learning models to make local decisions on who to replicate data to.

PoPs may also serve as caches for databases. For example, **DBProxy [5]** uses materialized dynamic views at PoPs, to provide low latency queries for relational database systems. At each edge location, partial table replicas are built based on observed queries. The tables' schema at the edge is dynamic, so tables can be modified to add more columns from the back-end's base table over time. Join tables are also built from the result of join operations with the same list of affected tables. By sharing common schemas (for join and base tables) as much as possible, storage space can be saved on schema-related overhead. Tables are initialized by first observing the query stream passing through a given PoP. For efficient query matching, PoP locations maintain indexes of previously executed and cached queries. When checking if an incoming query can be answered by locally cached data, previous queries which consulted the same tables and affected a superset of the incoming query's columns are identified. If the interval of values for the incoming query is contained within the results observed by previous queries, the query is answered from cache. As for consistency, all modifications to the back-end database are propagated and applied to the PoP caches, including rows which were not previously stored at those locations. Although not stated, we assume write transactions are either applied in a write-through manner to the caches, or all write operations happen exclusively on the back-end side. To limit the size of the cache, a background garbage-collection is activated when the occupied storage passes a given threshold.

As applications matured, PoPs started being used for other purposes such as Function-as-a-Service (FaaS) applications [72], or multimedia content streaming [108]. **Meiklejohn et al.** [72] present an interesting approach to database implementation on PoPs. The authors' system leverages FaaS platforms from big cloud providers to employ independent, but eventually consistent, database replicas at PoPs, for use in collaborative shared state scenarios. Placing non-blocking data replicas closer to the users provides increased throughput while also lowering latency. The example given is that of the popular tv show, Game Of Thrones, who's wiki page receives not only read but also many write operations in the hours surrounding a new episode's launch. The system uses CRDTs [100] to allow different replicas to perform concurrent and unordered writes while still ensuring an eventual convergence of their state. The platform chosen for deployment was Amazon's Lambda@Edge, a FaaS service that targets the execution of stateless and transient functions, which means providing persistency over such a system is not trivial. Since Lambda@Edge does not provide communication between lambda functions, an external AMQP broker is employed at the closest region's Amazon datacenter, so that data may be replicated across nodes, making this a fog computing system. Furthermore, the system ensures that lambdas in a given region are periodically invoked, to avoid losing updates due to inactivity induced deallocation of resources. The advantage of using Lambda@Edge, aside from placing data closer to users, is the elasticity

mechanism it provides, which automatically scales-out resources to the PoPs where they are needed.

Although not covered in surveyed solutions, **PoPs often have dedicated connections to central cloud datacenter infrastructures** of the same provider [7], offering benefits in bandwidth and network stability between infrastructures, which could be explored by future database systems. Furthermore, the **mechanisms provided within PoP locations are lacking in functionality** (e.g., no communication between functions), which difficults the implementation of databases in such environments. However, as PoP systems evolve, such mechanisms may become available.

2.3 Autonomous Vehicles

Autonomous Vehicles (AVs) can both generate and consume big amounts of data, so developing databases that are able to handle such data is especially challenging. On one hand, the data generated by onboard sensors is still largely unexplored, providing an opportunity for new data mining scenarios. In order to extract useful information from such systems, databases must manage massive amounts of generated data, take into account the biased nature of such information, and account for the complex privacy concerns that may arise from its use [56]. On the other hand, AVs may also require data stored outside the vehicle to ensure a safe driving experience, e.g., by receiving timely updates of unexpected events, such as crashes, which may affect them.

AV database systems By preemptively providing AVs with 3D maps of upcoming stretches of road, driving decisions can be made more safely. **3D-MADS** [49] uses RoadSide Units (RSUs) with short-range low-cost transmission capabilities, and Local Data Controllers (LCDs) which provide long rage, high cost cellular transmission capability to optimize the dissemination of 3D data. To lower the amount of exchanged messages, index coding is applied, so that 3D data for multiple vehicles is encoded into shared messages and relayed to all participants. Each one can then decode it by applying bitwise operations such as XOR between the received data, and the data that is already onboard. To optimize index coding, LCDs retrieve trip plans from all vehicles, optimizing for maximum RSU-based low cost transmission, and minimizing the use of costly LCD cellular data transmission. In order to keep maps updated, locally perceived data is uploaded from vehicles to LCDs and RSUs, after applying differential coding techniques which isolate the data elements that differ from the downloaded data. In the future, integrating index and differential coding directly into the DBMSs of RCUs and LCDs could lead to further performance improvements.

LiDAR data, 3D map data acquired by sensors in AVs, is stored in file formats such as LAS [51]. **Boehm et al.** [12] points out that companies invested considerable time developing file-based processing pipelines, but an exponential increase in amount of data, and requirements for redundancy, scalability and availability call for a shift towards database system adoption. As such, they propose the use of NoSQL document store solutions, such as MongoDB, to allow storage and processing of LiDAR data using efficient database systems, without incurring in a large logic conversion cost. Moreover, as MongoDB provides spacial indexes, it places itself as a great choice for storing this type of data, providing increased query performance.

Keeping track of vehicle location over time allows predicting traffic flow, avoiding accidents and improving traffic speed. Usually, DBMSs assume data to be discrete, only changing at specific points in time. This is not the case for moving vehicles, where their location is continually changing. **DOMINO** [121] proposes an efficient moving object representation for database systems. To keep the location of a car within an acceptable accuracy range in a (e.g., relational) database system, a big amount of updates would be required, thus incurring a large overhead. Instead, DOMINO proposes the MOST data model, an efficient representation of vehicle location based on position, direction of motion and speed data. Between updates, the location of vehicles may be

determined using inference, which provides a location with an associated degree of uncertainty. DOMINO also describes how spacial indexes can allow mapping vehicles to geographical regions and determining region overlap of different vehicles at a given time, enabling faster execution of queries such as: *determine the vehicles that will pass through a given intersection in the next 5 minutes.*

Efficient representation of vehicle position, and 3D data dissemination and storage have been studied separately, however, to the extent of our knowledge, **a full database system targeting AV data is yet to be developed**. Publications are already calling for complete AV querying and storage systems, pointing out the advantages provided by extending those systems to the edge [56].

2.4 Other

Stream processing systems modify real-time data to be later consumed by other services in their post-processed state. As edge stream processing systems increased in complexity, stateful processing, through data persistence and sharing of state between different stream processing functions, became a necessity. **Afetti et al.** [2] employs DBMSs in these systems to provide them with transactional and consistency guarantees. Often, shared state is accessed by a small set of functions that are placed in proximity to each other at the edge. Thus, Gupta et al. [46] advocates that stateful edge stream processing databases can benefit from the differential consistency characteristics of a local-quorum model, where strong consistency is only guaranteed within local edge clusters. Setting geographical areas of interest for data elements implies replicas within that area must ensure strong consistency, serving functions that share that state. Remaining replicas consider replication to be done asynchronously, serving as backup in case of geo-correlated replica failure.

DBMSs that integrate Machine Learning models eliminate the cost of data migration between DBMSs and ML platforms. **Smartlite** [65] is a DBMS supporting in-database ML inference for resource constrained nodes. It is implemented over ClickHouse due to its columnar organization of data, which helps speed up tensor operations. To overcome the limited computing capabilities of edge nodes in floating-point operations, SmartLite applies quantization to neural models. Furthermore, it also employs pruning of weights which are not relevant to the model, reducing its size, as well as its inference time. Under the assumption that many models share a common pre-trained model, and only fine-tune the last few layers, SmartLite avoids having multiple copies of the same common layers in-memory. As models' data is memory mapped and not all data can fit in memory, SmartLite also employs model scheduling to group the execution of models that share common layers, minimizing the amount of pages exchanged between memory and disk.

As a final note, although there have been multiple developments in novel types of DBMSs, such as graph, vector and spatial databases [45, 55, 104], none of them consider data storage in edge nodes, so understanding how they could benefit from the edge is still an unexplored research axis.

3 DISCUSSION OF KEY EDGE DATABASE DRIVERS

Latency is often described as the main advantage of providing services at the edge [5, 122], while handling constrained hardware resources is its main challenge [16, 124]. Thus, an holistic view of both aspects, i.e., analyzing concrete latency boundaries and hardware resources in the identified application domains, proves key to establish setup characteristics and consequently render architectural decisions. On the one hand, understanding the systems' latency performance in the same application domain assists researchers to scope their own results, and eventually establish base targets. On the other hand, knowledge of the hardware used is key to determine relevant evaluation setups. In cases where previous solutions are not publicly available, setup replication is essential for result comparison. Energy and privacy are also discussed, as even though they are mostly not considered in edge database systems, they often have a major impact over edge resources.

Since hardware setups and workloads adopted by surveyed publications are highly heterogeneous, our analysis is not intended to render a direct comparison between systems, but rather to highlight that heterogeneity, and provide a broad view of future research directions.

3.1 Hardware

Edge use cases hold drastically different hardware characteristics to those of cloud based, or conventional centralized database systems. Table 1 summarizes the hardware capabilities for the setups used in systems' evaluations, from which two patterns emerge.

The first pattern is characterized by: severely resource constrained nodes; asymmetric nodes, i.e., with differing capabilities; a hierarchical hardware architecture, where the number of nodes in each layer and their hardware resources are inversely correlated; battery powered devices at the least hardware capable layer(s); novel databases, that present themselves as new storage solutions; and high cluster sizes, in the order of hundreds to several thousands of nodes (as shown in Section 5.2). This pattern applies mostly to **Sensor Network systems**. The second pattern places server-grade machines at strategic points, in small numbers, and with symmetric resources, often adapting particular characteristics of existing DBMSs to the edge, instead of building an entirely new system. That is the case for the remaining use cases identified.

Distributed databases over in-network (SN-IN) and isolated sensor networks (SN-I) consider sensor nodes themselves as the storage devices, envisioning the use of very low capacity, and battery powered, nodes. For the case of SN In-Network systems, there is usually also a base station node, with more resources and a constant supply of energy, where query optimization is performed, before execution takes place in the network. Some exceptions however, such as Antelope [111], are able to execute queries entirely on the sensors themselves.

There are sensors with processing units as small as a Mica Mote [27, 125], with 1-core at 4Mhz and 4 KB of RAM memory, up to slightly better nodes with 16MHz single-core CPUs and up to 10 KB RAM. Persistent storage capacity shows a bigger variation, as lower powered sensor nodes have only 512 KB flash memory, as in Cougar [27, 125], while others have multi-gigabyte SD cards, as in Antelope [111]. There is no correlation between the time of publication, and the capacity of the nodes considered. E.g., Vasilescu et al. [113] use 16Mhz single-core CPU sensors and 4 KB RAM memory, in a work that dates to 2005, while TinyDB [69], in 2015 uses sensors with just 7MHz and also 4 KB of RAM. **This should be something to consider in future solutions**. In nodes capable of query optimization, the most powerful node surveyed uses a 400MHz single-core CPU with 64 MB of RAM memory [63].

Similarly, **sensor network search-engines (SN-SE)** share the same concerns and constraints in terms of hardware resources at the sensor level as SN In-Network and SN Isolated systems, with more powerful intermediate nodes being used for indexing. Dyser [83] is the exception, conducting its evaluation in a server, where sensor traces are replayed.

Sensor network sink nodes (SN-SN), or gateway nodes, are intermediate time-series databases located between end devices and the cloud. Sink nodes are one step above the hardware used for sensor nodes, using small form computer devices such as RaspberryPi devices or even low-end servers. Between 1 and 8 cores are available, with frequencies between 0.25-2.1GHz, 0.032-32 GB of RAM and few GB to TB of storage space. Despite these nodes having better resources than the previous use cases, they considerably fall below of cloud datacenters.

Point of Presence (PoP) works adopt the use of either Personal Computers (PCs) or sever-grade machines. In surveyed PoP solutions, only 2 provide hardware information [5, 67], and they do not detail all components of the system. Thus, it is not possible to establish a reference plane for this application domain based on just this information.

Autonomous vehicle (AV) works lack details for hardware resources [12, 49, 121].

Table 1. Hardware analysis

System	Use Case[1]	Year	Device	Cores	CPU Frequency	RAM	Disk
Wu et al. [122]	SN-SN(P)	2021	*Sensor:* nRF52840 *Edge-L1:* RPi 3B+ *Cloud:* DigitalOcean	1 4 –	64MHz 1.4GHz –	256KB 1GB –	1MB – –
Rahmani et al. [90]	SN-SN(P)	2017	*Sensor:* – *Edge-L1:* PandaBoard *Cloud:* –	1 2 –	1.6-32MHz 1.2GHz –	2-86KB 1GB –	32-512KB 8GB SD card –
ApacheIoTDB [115]	SN-SN	2020	*Sensor:* RPi [U]#1 *Edge-L1:* RPi [U]#1 *Cloud:* RPi [U]#2	1-4	0.25-1.5GHz	0.25-8GB	–
EdgeDB [124]	SN-SN	2019	*Edge-L1:* Server [U]	8	2.0GHz	32GB	RAID0 250GB SSD 8TB HDD
Respawn [16]	SN-SN	2013	*Edge-L1: ARM9* *Cloud: Server*	1 8	450MHz 1.86GHz	32MB 4GB	64GB SD card –
ModelarDB [54]	SN-SN	2023	*Edge-L1: PC* *Cloud: VM*	4 8	– –	4GB 32GB	HDD SSD
TorqueDB [40]	SN-SN	2020	*Edge-L1: RPi 4B* *Edge-L2: PC* *Cloud: MS Azure*	4 6 4	1.5GHz 2.1GHz 2.3GHz	2GB 8GB 16GB	64GB SD card 0.5TB HDD 0.5TB HDD
Antelope [111]	SN-IN	2011	*Sensor #1:* MSP430F1611 *Sensor #2:* Tmote Sky	1 1	3.9MHz 8Mhz	– 10KB	– 1MB + GB SD card
TinyDB [69]	SN-IN	2005	*Sensor:* Mica Mote 2	1	7MHz	4KB	128+512KB Flash
Cougar [27, 125]	SN-IN	2002/3	*Sensor:* Mica Mote	1	4MHz	4KB	512KB Flash
Presto [63]	SN-P	2009	*Sensor #1:* TelosMote *Sensor #2:* Mica Mote 2 *Edge-L1:* Crossbow Star Gate	1 1 1	8MHz 7MHz 400MHz	10KB 4KB 64MB	–
EnviroStore [68]	SN-I	2007	*Sensor:* PC #1 *Edge-L1:* PC #1	1	1.7GHz	1GB	12KB 64KB
Vasilescu et al. [113]	SN-I	2005	*Sensor:* Fleck *Edge-L1 #1:* 8-bit MCU *Edge-L1 #2:* Crusoe	1 1 1	16MHz – –	4KB 2KB –	640KB – –
Search NDNoT [98]	SN-SE	2017	*Sensor:* Arduino *Edge-L1:* PC	1 –	16MHz –	2KB –	2GB SD card –
Dyser [83]	SN-SE	2010	*Edge-L1:* Server	4	2.66GHz	–	–
Snoogle [117]	SN-SE	2008	*Sensor:* TelosB *Edge-L1:* TelosB *Edge-L2:* HP iPAQ	1 1 1	8MHz 8MHz 522MHz	10KB 10KB 64MB	1MB 1MB 128MB
Meiklejohn et al. [72]	PoP	2018	*Edge-L1:* AWS Lambda	–	–	–	–
SEQ/Hybrid [67]	PoP	2007	*Edge-L1:* PC	–	1.5-3.0GHz	0.5-2GB	–

DBProxy [5]	PoP	2003	*Client:* PC *Edge-L1:* Server *Cloud:* Server	1 1 1	200MHz 400MHz 1GHz	64MB 128MB 256MB	–
SmartLite [65]	Other	2023	*Edge-L1:* ARMv8	8	1.8GHz	16GB	–

Symbols: [*U*] Unspecified #X Device ID – Not Available

In overview, the amount of resources available at each edge node dictate the tasks they can perform, and impact on the architecture of the database solution, i.e., in which nodes (cloud or edge) developers will decide to place each computational function. For example, Cougar [27] proposes the use of a more powerful edge node to compute a query plan, which is then relayed to sensors for execution. This is done as the limited resources of sensor nodes are not enough for query planning. In contrast, a cloud-only database can use the same node for query planning and execution. We note that there are no publications sharing the same setup, nor do they evaluate alternative solutions using their setup, ultimately preventing them from producing comparable results. Also, some publications resort to hardware that is older and less capable than publications which postdate them.

Constrained hardware resources are one of the **major limitations of edge devices, directly affecting the design of edge database solutions**. Future publications should consider using **state-of-the-art hardware** to maximize the capabilities of edge nodes. Moreover, publications should consider **replicating setups from previous works in the same application domain, or running competing systems using their setup**, in order to make them comparable.

3.2 Latency

Both FogStore [46] and EdgeDB [124] use latency as one of their main motivations for moving storage to the edge. However, they do not provide concrete latency values showing that the difference in latency between a cloud and edge deployment can be the deciding factor for a system to perform adequately. For the publications which provide comparisons between edge-enabled and cloud-only deployments, it is hard to assess if the gap between them is meaningful. Furthermore, the latency requirements for applications making use of databases at the edge is not clear.

For web applications, latency requirements have increased over the years. In 2000, users considered a latency of up to 15s for web page retrieval to be acceptable [84]. By 2004, users were only willing to wait for 2s before abandoning a non-loading page [77]. Latency can also affect the dynamics in competing web services, as customers are more loyal to services with lower latency values [107]. Pages 250-500ms faster have a higher chance of a user clicking on them [6]. More interactive systems like map panning applications have a greater impact on latency requirements, with users expecting latencies as low as 100ms [15]. For search-engines, users are 1.5 times more likely to choose a system that has a query latency around 250ms. These findings are relevant for PoP and SN-SE applications, generally showing that web systems with lower latency provide both a competitive advantage and greater user satisfaction.

For Autonomous Vehicles, traffic events must be acted upon within 100ms [66]. If that requirements implies the exchange of data with data storage nodes at road-side units, or querying locally stored data, database systems must provide latencies even lower than that.

Requirements are better established for End-to-End latency of networks used for smart and IoT applications, with values ranging between 0.25-250ms for healthcare, gaming, transportation systems, education, smart grid and factory automation applications [21, 86]. For tactile internet applications (i.e., applications with tactile feedback), there is a 1ms requirement [103]. Placing data processing/storage nodes at the Edge (e.g. in cellular towers) is the solution to achieve such low

Table 2. Latency in Edge vs Fog vs Cloud

System	Use Case[1]	Year	Latency			OP Size/Nr
			Edge	Fog	Cloud	
Wu et al. [122]	SN-SN(P)	2021	11.5-2078ms	–	94.5-2233ms	10B
Rahmani et al. [90]	SN-SN(P)	2017	95ms	102.9-109.8ms	106.6-213.3ms	240KB
EdgeDB [124]	SN-SN	2019	1-512ms*	–	–	16B
Respawn [16]	SN-SN	2013	310ms (R) 26404s (W) 340ms (R)	– – 85ms (R)	23.9ms (R) 373s (W) –	1k-point 1M-point –
TorqueDB [40]	SN-SN	2020	473ms (R)	–	388ms (R)	–
Antelope [111]	SN-IN	2011	4-83ms local(R) 1-9s network(R)	–	–	1 query 1-10k queries
Presto [63]	SN-P	2009	2-30 s	–	–	–
Krentz et al. [58]	SN-A	2019	5-14ms (W) 81ms-4.33s (R)	–	–	– Variable size
Search NDNoT [98]	SN-SE	2017	1-10s	–	–	–
IoT-SVK [31, 32]	SN-SE	2012	20-50ms (S)	–	–	–
Dyser [83]	SN-SE	2010	200-260ms	–	–	–
Snoogle [117]	SN-SE	2008	30-90ms (W) 100-260ms (R) 4.6-6s (R),[E]	–	–	20-160B 4-16B 4-40B
SEQ/Hybrid [67]	PoP	2007	45-180ms (R) 20-220ms (W)	–	–	–
DBProxy [5]	PoP	2003	–	263-540ms	385-1024ms	–
3D-MADS [49]	AV	2019	9-17ms (E2E)	–	–	1024 B
Boehm et al. [12]	AV	2015	0.268s (R) 1.515s (W)	–	6.2s (R) 95.8s (W)	40MB
FogStore [46]	Other	2018	60-70ms (R) 40-90ms (W)	–	–	–

Symbols: (E2E) End-to-End (R) Read (S) Search
[E] Encrypted (W) Write * Single machine test

values between the data source and processing/storage location. Hence, although these latency requirements are not directly aimed at database systems, they may require database-enabled applications to be deployed at the edge instead of the cloud, impacting all depicted use cases.

Some publications evaluate latency requirements for sensor based applications [20], placing requirements between 105-2700ms, with the maximum lowering to 600ms when considering only optimal scenarios. Nonetheless, those applications do not consider the use of edge databases.

Concrete requirements for latency in edge applications are scarce, and they do not specifically target database systems, making it harder to scope if the latency advantage provided by the edge translates to a real benefit for edge database system users.

Table 2 summarizes latency values in surveyed proposals. It is important to note that the methodology followed across different solutions can be drastically different, e.g., due to differing setups (see Section 3.1) and size of operations measured. Thus, latency results from different publications are not comparable, even amongst solutions of the same application domain. Notice also that none of the works directly compare their performance values with those of their competitors, making it hard to gauge if new systems are actually providing an advantage over state-of-the-art. For

example, ApacheIoTDB [115, 116] does not compare its latency/throughput performance against Respawn [16], or other SN-SN systems. However, it is possible to form a birds-eye view of latencies being observed in edge, fog and cloud scenarios, and most importantly, measure the magnitude of difference between a cloud-only and a fog or edge computing scenario.

In this survey, only 22% of systems compare the latency values of edge/fog scenarios against cloud scenarios. Furthermore, some only consider End-to-End latency, i.e., the time it takes for requests to traverse a network, which provides no significant insight to the latency value of the proposed database. When comparing the best latency scenarios between edge/fog environments and cloud environments, the edge/fog deployments show, on average, 41% lower latency. It is not yet possible to evaluate the latency difference between edge and fog deployments as only one publication compares them. This paper excludes from average calculations the results of Respawn [16], which considers a deployment scenario with severe network and resource limitations at the edge. Contrary to other systems, its objective is to offload as many aggregated values from edge to cloud as possible, since only the cloud has resources to answer queries with low latency. In the case of Respawn, read-only queries are up to 12.97 times slower on the edge than on the cloud.

Absolute values are reported by a small fraction of publications. From those that do, edge systems show an average latency of 332ms, fog systems 150ms, and cloud systems 1.2s, considering best case scenarios, but excluding Respawn [16] and Boehm et al.'s 95.8s for cloud writes [12], as it is an outlier. These values show great advantage in latency for edge and fog computing; and point to a need for edge/fog deployments to be able to meet the requirements previously established.

Most works lack latency comparisons between edge and cloud based deployments, which makes quantifying the advantage provided by the edge's proximity to the end-devices harder. Furthermore, there is a **lack of latency measurement methodology**, which prevents comparing the latency performance of solutions targeting the same application domain.

3.3 Energy

Energy limitations imposed on edge database systems can be analyzed through two questions:

[Q1] What are the energy limitations faced by edge database nodes?

[Q2] What energy optimization mechanisms do edge databases adopt?

Energy limitations (**[Q1]**) dictate how much energy is available for use by database nodes. Nodes may have: a finite amount of energy (e.g., a battery); an amount of energy limited in time (e.g., solar-powered); or a virtually unlimited amount of energy, excluding grid resilience issues (e.g., mains powered). Energy optimizations employed by the edge database systems (**[Q2]**) dictate how the database is able to meet those requirements.

[Q1] In terms of energy limitations, two scenarios are prevalent in edge database systems.

- **Battery powered sensor nodes.** The amount of available energy is inherently limited. [27, 63, 68, 69, 90, 91, 101, 111, 127].
- **Mains powered devices.** Nodes are powered by mains electricity, so power limitation is not a concern [5, 67, 83, 98, 124].

In scenarios where available energy for a given time-interval is limited, Edge devices may resort to energy harvesting mechanisms (e.g., solar or piezoelectric energy source) [25, 44] to directly power those devices, or to replenish a battery's charge. In those cases, the objective may be to adopt an elastic energy budget approach, where a trade-off between energy consumption and performance occurs according to current energy production and remaining stored energy. Only one of the edge databases, Presto [63], considers energy harvesting, where proxy nodes can be

solar-powered. However, it considers proxies to have an unlimited energy supply, and does not employ any energy optimization at the proxy node level.

[Q2] Energy optimization adoption is scarce and employed only for battery powered sensor nodes. Optimizations are prevalent in the SN-IN, SN-P and SN-I use cases, as they consider the sensors themselves to be the main storage nodes in the database system [27, 63, 68, 111]. They focus on reducing the number of messages exchanged, as it has been shown that data transmission has a larger energy impact, when compared to data processing or storage [111]. SN-IN systems reduce communication by pre-computing query plans at mains powered sink nodes which limit the list of contacted sensors to those containing data relevant to the query, and by reducing message size through aggregation operations that occur as data is relayed between sensors until it reaches the sink node [27]. Sleep schedules are also optimized to reduce active time. Sensor sampling rate can be adjusted to ensure that sensors have enough energy to fulfill long-running queries [69] (see *In-Network Processing Architecture (SN-IN)* in Section 2.1). For the case of Antelope [111], full DBMS deployment at each sensor ensures no data other than that explicitly queried by a user leaves the sensor, while adopting techniques such as FLASH access pattern optimization provide further energy savings. Alternatively, SN-P systems [63] use machine learning models to reduce communication. Sensors are only contacted while models at the sink node are being trained, or when the predicted values deviate significantly from the real values being measured, in which case the models are updated (see *Model Prediction Architecture (SN-P)* in Section 2.1). SN-I systems do not have a constant connection to a centralized storage location, and often must store data for long periods of time, until a data mule or other means of offloading data is available. One interesting case is that of EnviroStore, which must balance between exchanging data with neighbors to avoid local storage overflow, and minimizing data communication to lower energy consumption [68] (see *Isolated Architecture (SN-I)* in Section 2.1). In rare cases, SN-SN systems also consider sensors' data format as a source of energy efficiency [90]. In sum, sensor energy optimizations focus on: minimizing the number of messages exchanged, reducing message size, optimizing FLASH storage usage, optimizing rate sampling and maximizing sleep time.

There are also exceptional cases that consider energy optimizations without an inherent need for them, as the setups they describe are powered by mains electricity [98]. The rest of the systems, which account for the majority of publications evaluated, do not consider energy consumption, or acknowledge the energy limited nature of edge nodes but do not explore it [5, 12, 13, 16, 31–33, 40, 46, 49, 54, 58, 65, 67, 72, 83, 85, 87, 115–117, 121]. These include all systems from the SN-SE, PoP and AV use cases, as well as most of the SN-SN systems, where sensors play a passive role collecting metrics and relaying them in their raw format to sink nodes.

It is clear that energy consumption is of concern for edge database systems, particularly for sensors, as they are often powered by batteries. How and when the batteries of sensors are replaced, and whether knowledge of such questions could be included in query optimization is not explored. Furthermore, the energy limitations of other edge nodes such as Sink Nodes, Edge Gateways, PoPs and Roadside Units, amongst others, are not well understood, as publications assume them to be powered by mains electricity. Surveyed publications do not clarify if there may be other energy limitations at play, such as the possibility for the infrastructure that supports them to have a limitation on the amount of power it is able to provide, or if energy harvesting is possible.

Energy limitations of edge database systems are not well understood. Publications should **evaluate the energy limitations present in real edge database deployments** to identify which limitations may be at play for Sink Nodes, Edge Gateways, PoPs, Roadside Units, Autonomous Vehicles, and similar nodes. The same question applies to understand how the energy in battery powered sensors is replaced. Furthermore, future **SN publications outside the SN-IN domain**

should consider optimizing the data retrieval process from sensors, by considering strategies employed in SN-IN systems to prolong the lifetime of the sensors' batteries.

3.4 Note on Security and Privacy

Privacy at the edge holds two perspectives. On the one hand, edge devices present a security disadvantage, as they are deployed in locations with greater physical exposure, and their often limited resources could compromise the quality of algorithms used. On the other hand, they present an advantage from the data placement perspective. Multiple layers holding data at increasingly shorter distances from data sources allow data to be stored closer to its origin, and with different degrees of data aggregation between layers. Thus, the edge can help answer data regulations restricting the location of data, as well as capacitate data owners with greater control over how their data is handled. Below we take a closer look at each of these perspectives.

Security. End-devices in many edge applications carry detailed, and sensitive information. E.g., smart meter information may allow attackers to infer various living habits, such as current TV channel, and specific audiovisual content being watched [42]. Nonetheless, only six surveyed solutions incorporate or mention the use of authentication and encryption mechanisms [68, 90, 98, 111, 117, 122]. Furthermore, hardware limitations and the increased physical exposure of devices to possible attackers present increased difficulty for ensuring the security of data being received, processed, and transmitted on these devices.

Future research should **consider authentication and encryption mechanisms as a core element of their designs**. Furthermore, the **trade-off between computational efficiency, and the resilience of authentication and encryption solutions**, in edge database environments, should be better understood.

Data placement. Data regulations converge towards restricting what and where data can be kept (i.e., personal data), especially in corporate and governmental scenarios. Such restrictions require data to be stored closer to the origin of data. For example, European regulations prevent data from citizens from being kept outside EU boundaries [82]. An edge-enabled system with resources placed closer to the source of data (i.e., at different locations) can provide a way to meet such regulations. Moreover, exploring aggregated representations of data at layers further away from the data source can help remote consumers achieve better latency performance, as long as the aggregated data form is sufficient to answer requests. Complying with data regulations such as the GDPR [112] is currently a requirement in some geographies, enforcing the right of individuals to understand how their data is being processed, and to opt out of sharing their information with 3rd parties. As such, exploring edge-enabled multi-layer systems can be the next step towards providing individuals with greater control over their data, namely, through data placement preferences.

From surveyed solutions, only three mention the data placement advantage provided by the edge [40, 105, 111], and they do not integrate that advantage into their solutions.

Future research must consider the **geographical distributed nature of edge nodes as an advantage of edge-enabled systems, which provides greater control over data placement**, helping deal with increasingly strict data regulations, and the increasing concern with the right to individuals' control over the data they produce.

4 EDGE DATABASE ARCHITECTURES

Most relational cloud database systems share common architectural similarities, such as depicted in Figure 2. Edge database systems reuse the same components, but depending on the application

domain: components may be absent; a component may be split between multiple nodes; or components may be implemented differently, leading to novel architectures. This chapter analyses the role of each component, how the architectures from each application domain differ from the cloud reference architecture, and in what components each application domain focuses on.

As depicted in Figure 2 (based on [48]), a database holds 4 core components: Client Communications Manager, Query Engine, Storage Manager and Replication Service. A description follows.

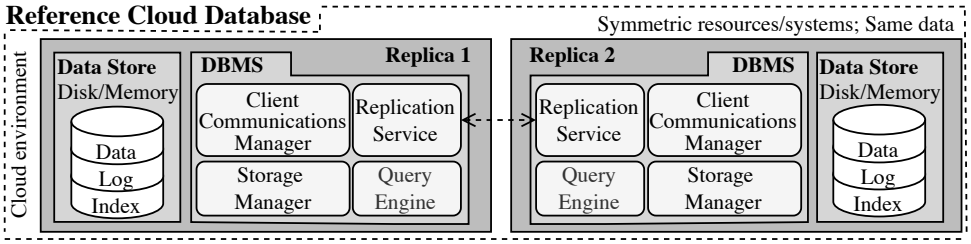

Fig. 2. Cloud database architecture - Illustration is a simplification of the architecture proposed in [48]

Client Communications Manager Exports well-defined client interfaces to safely and efficiently deliver requests to the database and retrieve results. Usually at least one code oriented interface is available, e.g., Java Database Connectivity (JDBC) API, for communication between the application and data tiers, as well as a terminal-based client interface for administration purposes.

Query Engine Computes an execution plan for each query, assessing the cost for each stage of a query's execution. Statistics collected from the stored data are used to compute and decide the best execution plan for a query. The plan is then executed by scanning, filtering, sorting, projecting, grouping and applying other processing tasks over data items.

Storage Manager Stores and manipulates data at the Data Store. Manages various data structures, which vary according to the particular type of database, but for a relational database can include data tables, indexes, and a Write-Ahead Log. Although not all databases provide ACID properties, or transaction support, those that do, do it at the Storage Manager level. Using techniques such as copy-on-write and/or locking mechanisms, it ensures that, e.g., no two concurrent transactions see an inconsistent view of data, and that at commit time only a single concurrent transaction is able to succeed, while others are aborted. It may also manage data buffers, moving data between disk and memory, in order to provide the best performance, or concern itself with data ordering, a common concern for, e.g., time-series databases [116].

Replication Service Distributed database systems are an evolution of monolithic systems, spreading the capability to serve and query data along a set of (often) identical nodes. The cornerstone of such engines depends on the capability to split or replicate data partitions, allowing to scale query execution to a growing number of database instance nodes. Running multiple replicas also ensures high-availability and fault-tolerance, i.e., the capability to continue to answer requests, even when a pre-defined number or percentage of replicas stops functioning correctly. This service handles replication of data between DBMS instances, ensuring that consistency constraints are met, e.g., causal consistency [59]. The replication service may also manage shard constraints, deciding how to split data into different partitions and how to allocate them between instances.

Within each application domain, advancements to the state-of-the-art tend to focus on a particular subset of the components, which usually constitute the bottleneck of the solution.

Sink Node Sensor Network DBMSs (SN-SN), whose architecture is depicted in Figure 3, merge data from many input sensor streams, through sink nodes, into single points of data storage,

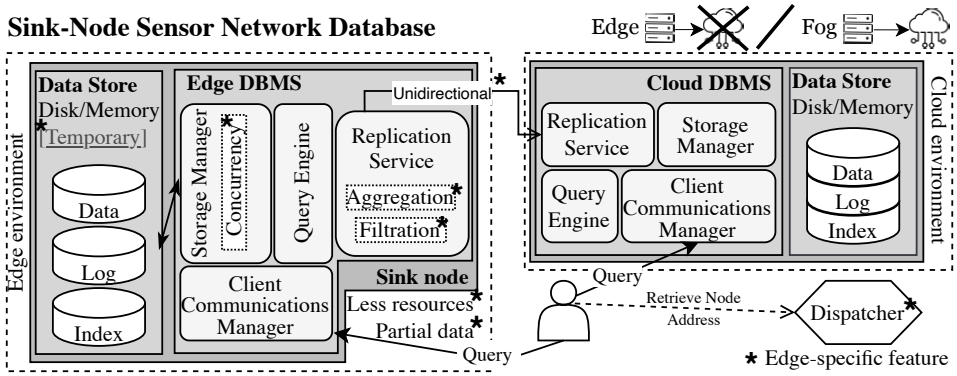

Fig. 3. Architecture for SN-SN databases

usually a cloud database. These DBMSs are composed of multiple (resource limited) sink nodes and a (more capable) cloud database. Each sink node aggregates data from a subset of sensors, pre-processing data, and then uploading it to the cloud. Clients can query both sink and cloud nodes, so a dispatcher may be required to redirect clients to the correct node [16]. Since sink nodes often rely on flash storage (e.g., SD cards), SN-SN DBMSs improve on **Storage Managers** by designing them to handle large numbers of parallel input streams, avoid wear of particular flash-storage cells, and maximize throughput by means of buffering and batched write operations [115, 124]. To cope with elevated data arrival rates, most recent data may reside entirely on memory [16], sacrificing durability guarantees for throughput. In parallel, **Data Stores** are optimized to deal with limited storage, e.g., by adopting specialized memory limited indexes [33, 85, 124], or by maintaining data for a certain period, deleting it once it has been replicated to a centralized storage location [115]. Contrary to cloud nodes, **replication** in SN-SN DBMSs tends to be unidirectional, with data being migrated exclusively from the sink nodes to the cloud database. Moreover, data tends to be filtered, aggregated and/or compressed before being replicated, especially when bandwidth between edge and cloud nodes is limited [16, 115, 124]. Sink nodes store raw data as it was captured, but only for assigned sensors, while cloud databases keep pre-processed data for all sensors. One notable exception to this representative architecture is TorqueDB [40], which by splitting the query engine between asymmetric nodes, and storing data in a decentralized way presents an architecture which is much closer to that of the SN-IN systems described below.

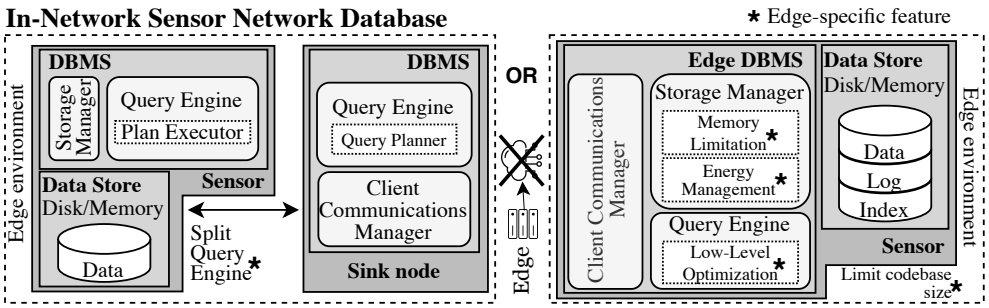

Fig. 4. Architecture for SN-IN databases

In-Network processing DBMSs (SN-IN), make sensors themselves an integral part of the system, adding to data generation the duties of data storage and query execution. Systems tend to follow one of two architectures, as shown in Figure 4. In the first, the database is split between sink nodes and sensors, with the major difference from cloud databases being the distributed **Query Engine**. Query plans are pre-computed at sink nodes, which are non-battery powered and have greater processing capacity, with the objective of minimizing the number of contacted sensors, and of exchanged messages between them [69], promoting better energy-efficiency for battery powered sensors. At the same time, they orchestrate sensor sleeping schedules that allow inter-sensor communication with minimum wake-up time, further improving energy expenditure [27, 125]. In the second, each sensor runs an independent DBMS, with sensors being queried individually. To do so, the DBMS must take into account the extremely limited capabilities of these nodes, i.e., single core CPUs with less than 1GHz clock speed, and memory in the order of 1 MB (see Section 3.1). Contrary to cloud systems, the **Storage Manager** focuses on minimizing energy and memory consumption, with systems achieving memory footprints as low as 4 KB. Furthermore, the **Query Engine** is restricted to a subset of simpler operations, and subjected to cumbersome optimizations such as being executed on a custom virtual machine developed in bytecode to improve performance. In cloud-only systems, query plans target minimum latency, and maximum throughput, but in sensor databases, limited energy is also a concern since they are often battery powered. The Query Engine may opt for the plan that requires fewer messages to be exchanged between sensors to save energy even if that means an increased query latency [111]. The size of the entire codebase is also minimized to save on available space.

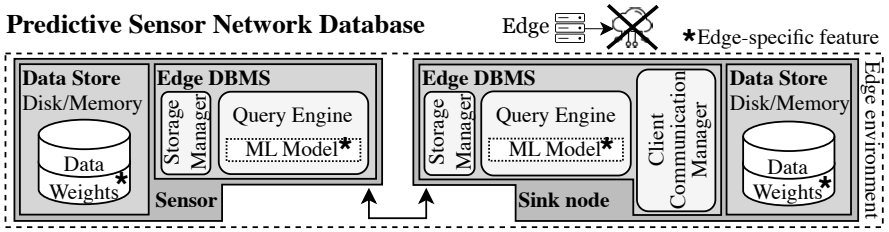

Fig. 5. Architecture for SN-P databases

Predictive Sensor Network DBMSs (SN-P) add a prediction model to the **Query Engine**, minimizing the number of messages exchanged with sensor nodes (Figure 5). Sink nodes use model predictions to answer client queries, while relying on sensor feedback to keep models updated [63]. This unique feature caters to the limited bandwidth in edge environments, and minimizes sensor battery consumption associated to message transmission.

Isolated Sensor Networks (SN-I) operate without a constant connection to neither sink nodes nor a cloud database. Thus, sensors must store data locally, until a data mule offers the chance to offload the acquired data (Section 2.1). As sensors may generate data at different rates, data redistribution is employed to avoid exhausting local storage in busier nodes [68]. Redistribution must also be energy aware, to balance data loss due to limited storage, with data loss due to battery exhaustion. In the cloud, replicas generally have identical storage capacity; individual nodes have more resources, leading to smaller cluster sizes; and nodes are not battery powered, thus implying less coordination overhead, and redistribution techniques that do not need to take energy into account. This architecture is depicted in Figure 6. SN-I databases focus on **Client Communications Managers** that offer efficient communication between sensor nodes, and between them and data

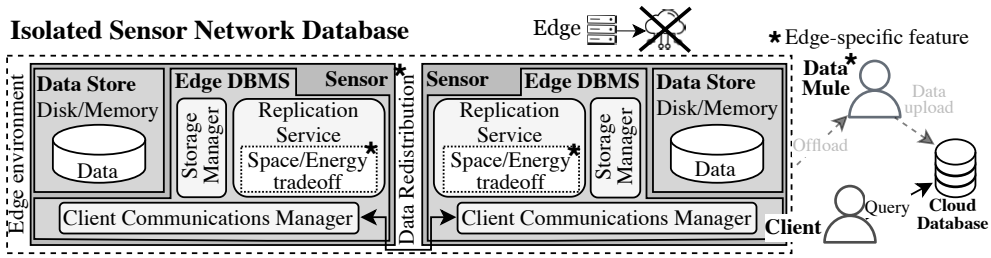

Fig. 6. Architecture for SN-I databases

mules, for data-offloading and redistribution. In parallel, the **Replication Service** is essential to determine when to offload data from one sensor to another, and which data to offload [68, 113].

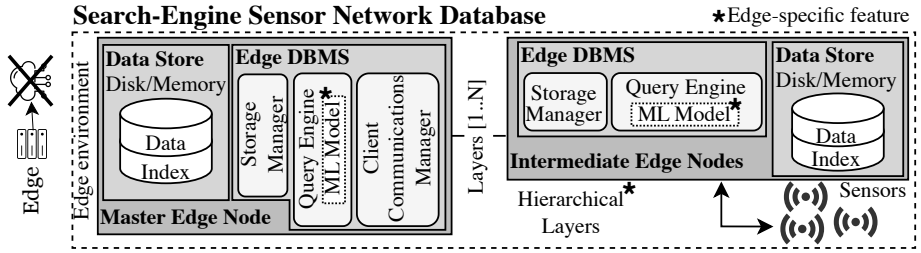

Fig. 7. Architecture for SN-SE databases

Search Engine Sensor Networks (SN-SE) develop **Query Engines** that specialize in retrieving the state of sensors associated to a textual query in the smallest interval of time possible. Efficient distributed querying over multi-layer, and often hierarchical, indexing structures [98, 117], and modeling of sensors' state patterns at gateway nodes [83] minimizes the number of contacted sensors for a given query (Figure 7). **Data Stores** employ special indexing techniques that allow multiple types of textual queries, i.e., spatial-temporal, value-based and keyword-based [32]. Data may be stored at the sensors, with edge nodes maintaining indexes to quickly access relevant sensors, or it may be kept at the edge nodes through updates sent by sensors as their state changes.

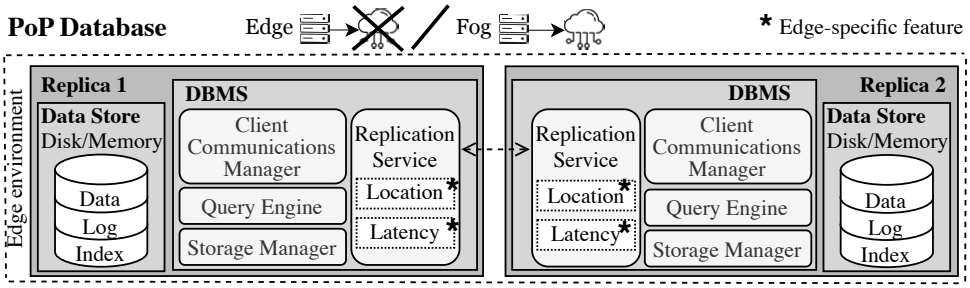

Fig. 8. Architecture for PoP databases

Point-of-Presence (PoP) databases focus on providing improvements at the **Replication Service** level while retaining the low latency advantage of the edge, a substantial challenge for

geo-replicated databases. Solutions are able to ensure strong consistency and complete replicas on all nodes, while ensuring write transactions only require a single system-wide Round-Trip-Time message exchange, ensuring minimum write-request latency [67]. In systems where edge nodes are used to cache database contents closer to users, but edge nodes contain limited storage space, replication services must decide which data to cache at each given moment [5]. In scenarios where clients are considered to be mobile and data is partially replicated, minimum latency can only be achieved if the data relevant to the client is replicated to the closest edge nodes [87]. Some systems leverage already available services running on PoPs to ease database development, i.e., leveraging self-scalable mechanisms of function-as-a-service platforms [72]. Figure 8 highlights the common architectural components of PoP systems.

Autonomous Vehicle (AV) systems have two main challenges. The first is data representation in the **Data Store**. 3D maps occupy a large amount of storage space, so efficient representation and compression of data is of concern [12, 49]. Moreover, representation of moving vehicles imposes the need for novel data formats which differ from current static oriented ones [121]. The second is in efficient communication of voluminous data in wireless environments [49], through the **Client Communications Manager**. Techniques such as index coding and differential coding allow multiple vehicles to be updated with a single transmission, and diminish the amount of data to be transmitted. As AV work is still in its early stages, a reference architecture is yet to emerge.

Table 3. Internal database components targeted by surveyed works

	Use Case [1]	Year	Client Comms. Manager	Query Engine	Replication Service	Storage Manager	Data Store
VergeDB [85]	SN-SN	2021	–	–	–	●	●
ApacheIoTDB [115]	SN-SN	2020	●	●	●	●	●
EdgeDB [124]	SN-SN	2019	–	–	–	●	●
Respawn [16]	SN-SN	2013	●	–	●	●	●
ModelarDB [16]	SN-SN	2023	–	●	–	●	●
TorqueDB [40]	SN-SN	2020	–	●	–	–	●
Antelope [111]	SN-IN	2011	●	●	–	●	●
TinyDB [69]	SN-IN	2005	●	●	–	–	–
Cougar [27, 125]	SN-IN	2002/3	●	●	–	–	–
Presto [63]	SN-P	2009	–	●	–	–	–
EnviroStore [68]	SN-I	2007	–	–	●	–	●
Vasilescu et al. [113]	SN-I	2005	●	–	–	–	–
Search NDNoT [98]	SN-SE	2017	–	●	–	–	–
IoT-SVK [31, 32]	SN-SE	2012	–	–	–	–	●
Dyser [83]	SN-SE	2010	●	●	–	–	–
Snoogle [117]	SN-SE	2008	–	●	●	–	●
Pfandzelter et al. [87]	PoP	2021	–	–	●	–	–
Meiklejohn et al. [72]	PoP	2018	–	–	●	●	–
SEQ/Hybrid [67]	PoP	2007	–	–	●	–	–
DBProxy [5]	PoP	2003	●	–	●	–	–
3D-MADS [49]	AV	2019	●	–	●	–	–
Boehm et al. [12]	AV	2015	–	–	–	–	●
Domino [121]	AV	1999	–	–	–	–	●
FogStore [46]	Other	2018	–	–	●	–	–
SmartLite [65]	Other	2023	–	●	–	●	●

Symbols: ● Focus – Not a focus

Table 3 summarizes the internal components targeted by each of the surveyed proposals. In conclusion, **internal database component development** for edge systems depends on **hardware constraints**, **deployment architecture**, and **network limitations**. In the future, **privacy regulations** may also play an important role in the development of edge databases.

5 EVALUATION APPROACHES

Edge-enabled databases require specific benchmarking tools, simulation tools, workloads, and datasets, as their application domains vary considerably from cloud-focused solutions. Furthermore, the scale and complexity of edge systems, e.g., sensor networks, and autonomous vehicles, make evaluating edge-enabled systems especially challenging.

Many of the surveyed solutions do not compare their performance with solutions of the same application domain, and although a few publications benchmark database performance at the edge, they only evaluate cloud-oriented databases running on edge nodes [28, 43, 70]. They do not consider edge-specific databases. Thus, we identify a clear lack of evaluation methodology.

5.1 Simulation and Benchmarking Tools

Since some application domains deal with large deployments, or with nodes which are not readily available (e.g., sensor networks, and autonomous vehicles), simulation and emulation scenarios are common for such application domains (Table 4). Simulation tools create partial representations of systems, by capturing particular behaviors which are relevant to the evaluation being conducted. For example, a sensor can be simulated by periodically sampling values from a distribution which is representative of the phenomenon being measured by the real device. In this case, details such as data communication and network instability are not considered. Emulation tools, on the other hand, mimic the entire hardware and software features of a system, translating all aspects, e.g., communication, processing, and storage, to a computer program. Overall, simulation tools are able to represent more nodes than emulation tools (see Table 5), but at the cost of lower fidelity.

In the subject of edge database systems, most works which consider such tools rely on network focused simulation tools, i.e., MaxiNet [120], LoRaSim [114], and TOSSIM [61], which limit the accuracy of evaluation results for database systems, since the storage component is not the focus of the simulation. However, some works use more realistic tools, namely MSPSim [36] for emulating sensors; and Autoware [38] for simulation of autonomous vehicles.

The sequence of operations that systems are submitted to during evaluation, i.e., workloads, can be categorized into: **simple, domain specific, traces, and field tests** (Table 4).

Simple workloads exercise databases' write and read operations, without following a pattern that is specific to the targeted application domain. The latter would be the case of a *Domain Specific* workload. So, whereas, e.g., in a sensor network system, a simple workload would request the value associated to a particular timestamp, a domain specific workload would instead issue aggregation queries, which are more common in real scenarios [115, 124]. Most of the analyzed evaluations, i.e., 11 out of 27, use simple workloads. On the other hand, 6 of them use domain specific workloads.

Trace workloads capture a set of operations submitted to a production system, for a given period of time. The captured operations are then submitted to the system under test in the same way they were to the production system. Since access to a production deployment is needed to obtain traces, and there is a risk that the data capture can affect the system, or that the data captured is bound by confidentiality, the number of such workloads available for use is very limited. Only 5 of the 27evaluations use trace workloads. In the case of sensor network systems, traces only contain sensor readings, and so they do not capture queries posed to the system by users. In such cases, query workloads are synthetic, detracting from the faithfulness of the evaluation.

Table 4. Evaluation analysis

System	Use Case [1]	Year	Node (E/Si)mulation Used?	Tool	Workload
Wu et al. [122]	SN-SN(P)	2021	M	LoRaSim [114]	**R** - S *
Rahmani et al. [90]	SN-SN(P)	2017	N	–	**R/W** - FT *
Bonci et al. [13]	SN-SN(P)	2017	N	–	**W** - T **R** - DS
VergeDB [85]	SN-SN	2021	N	–	**W** - S
ApacheIoTDB [115]	SN-SN	2020	N	–	**R/W** - FT
EdgeDB [124]	SN-SN	2019	N	–	**W** - T **R** - DS
Respawn [16]	SN-SN	2013	Y	Custom	**W/R** - FT/T
TorqueDB [40]	SN-SN	2020	N	–	**R** - DS
Antelope [111]	SN-IN	2011	M	MSPSim [36]	**R/W** - FT/S
TinyDB [69]	SN-IN	2005	N	–	**W** - FT/DS **R** - S
Presto [63]	SN-P	2009	N	–	**W** - T/FT **R** - S
EnviroStore [68]	SN-I	2007	Y	TOSSIM [61]	**W** - S
Vasilescu et al. [113]	SN-I	2005	N	–	**W** - FT
Search NDNoT [98]	SN-SE	2017	N	–	**R** - FT
IoT-SVK [31, 32]	SN-SE	2012	–	–	**R** - S
Dyser [83]	SN-SE	2010	N	–	**R** - S
Snoogle [117]	SN-SE	2008	N	–	**R/W** - FT
SEQ/Hybrid [67]	PoP	2007	N	–	**R/W** - DS [B] (TPC-W [109])
DBProxy [5]	PoP	2003	N	–	**R/W** - DS [B] (TPC-W [109])
3D-MADS [49]	AV	2019	N	–	**R** - T
Boehm et al. [12]	AV	2015	N	–	**R** - S
FogStore [46]	Other	2018	Y	MaxiNet [120] +Docker [73]	**R/W** - S [B] (YCSB [24])
SmartLite [65]	Other	2023	N	–	**R** - S

Symbols: **R** Read, **W** Write; S Simple, T Trace; * Entire system tested, not DB specific, DS Domain Specific; [B] Benchmark, FT Field Test

Finally, *Field Tests* forgo the use of recorded or generated operations, relying instead on deployments of real devices. Users (i.e, researchers) then submit operations to the system as if it were a real deployment. That is the case for 9 of the considered systems. The field tests considered have been conducted only in small scales, and with users posing only simple queries to the system.

Workloads can be aggregated and provided as an external evaluation tool, called a **benchmark**. These tools can be useful for producing comparable evaluations, as the same tool can be used to submit the same set of operations to two different database systems. SEQ/Hybrid [67] and FogStore [46] resort to known database benchmarks (TPC-W [109] and YCSB [24]), which use workloads representative of behavior of production environments. However, the environments targeted by those benchmarks are not edge specific, making these tools inadequate.

Benchmarking solutions. Unfortunately, nearly no edge specific benchmarking solutions exist, and there are even less such solutions specific to edge storage evaluation.

IoTBench [60] is an edge specific benchmarking tool that offers seven testing scenarios related to Vision, Speech Recognition, and Physiological Signal Processing. This tool allows the evaluation of *Edge-only* scenarios for data processing at edge locations (e.g. compression, signal enhancement), however none of the considered test scenarios approach data storage at the edge.

Defog [71] allows evaluating systems in *Cloud-only*, *Edge-only* or *Fog* deployments, focusing on latency evaluation. Furthermore, in the *Fog* configuration, multiple different task allocation schemes between edge and cloud nodes are possible. From the six different workloads proposed, only iPokémon, a *Pokémon Go-like* scenario, considers storage at the edge, showing considerable latency improvement for the fog variant when compared to the *Cloud-only* deployment. However, the measured performance is not specific to the edge storage component.

TPCx-IoT [110] is a benchmark solution for evaluating SN-SN databases that adopts TPC's *Express* model, adopting a simpler benchmark specification. It measures the number of operations processed by the database per second. This is the only benchmark solution that specifically targets edge storage evaluation, but it is not used by any of the analyzed publications. Its underlying workloads are based on YCSB benchmarks, which although useful for throughput analysis under different workload distributions, do not necessarily mimic workloads generated in edge scenarios.

Future research should focus on developing edge specific storage evaluation benchmarks and workloads. More specifically, benchmarks should cover a broader set of metrics (e.g., energy expenditure), recurring to **more realistic workloads** (e.g., based on the real traces), and be able to **compare *Cloud-only*, *Edge-only* and *Fog* deployments**. **New simulation/emulation tools targeting edge databases** are also needed to allow for more accurate evaluation results.

5.2 Scale

Evaluation setups should ideally be representative of scenarios identical to the ones a production system would encounter. For edge databases, two key components in defining such a setup are the number of nodes considered (scale) and how they are distributed between layers (deployment). It is important that both of these properties mimic production environments as close as possible. Table 5 summarizes the scale and deployment configurations of each evaluation setup. Overall, systems consider a maximum of three layers that hold data, e.g., Snoogle [117]. The timespan to which chosen datasets equate varies significantly, from hundreds of ms to the equivalent of a year of data, even within publications for the same use case, which constitutes evidence of a **missing standard evaluation methodology**. The same conclusion can be reached for the amount of operations considered. Furthermore, in most cases where traces are used, they are not replayed at the same rate the data was generated, and thus do not mimic the actual scenario those traces were captured in. They are instead inserted at the maximum rate a system is able to handle, in order to test throughput capabilities. Future work should take this into account.

Looking specifically at Sensor Network systems, the maximum number of real sensors used is 50. For simulated scenarios, only 36 sensors were considered, and with **severe limitations on the detail at which they were simulated** [68]. Traces consider up to 10 000 nodes, and emulations only 40. Furthermore, in terms of the amount of records being managed, the evaluation of the considered publications considers at maximum millions of points spanning several hours or even a year. Meanwhile, real industrial deployments can reach the scale of billions of data points per day [47], with many thousands of sensors. This does not mean that the proposed solutions cannot cope with this scale, (e.g. ApacheIoTDB can, as mentioned in [47]), but the evaluation done in academic publications does not reflect this. Between the subdomains within Sensor Networks, Sink Node (SN-SN) systems are the ones with the biggest evaluation scale, followed by In-Network (SN-IN) systems' evaluation. Sensor Network Search-Engine (SN-SE) systems do not conduct evaluations

Table 5. Scale and Deployment Configurations of evaluated publications

System	Use Case [1]	Year	# Devices	# Operations	Op size	Duration
Wu et al. [122]	SN-SN(P)	2021	*Sensors:* 1-50 ○ *Edge-L1:* 1-10 ○ *Edge-L2:* 1 ● *Cloud:* 0-1 ●	2-120k	–	30 mins - 10 h
Rahmani et al. [90]	SN-SN(P)	2017	*Sensors:* – ○ *Edge-L1:* 1 ● *Servers:* 1 ●	2-120k	–	8h
VergeDB [85]	SN-SN	2021	*Sensors:* –(G) ○ *Edge-L1:* 1 ●	400M	64 bit	–
ApacheIoTDB [115]	SN-SN	2020	*Sensors:* 2 ○ *Edge-L1:* 1 ● *Server:* 1 ●	–	–	–
EdgeDB [124]	SN-SN	2019	*Sensors:* 1-1024(T) ○ *Edge-L1:* 1 ●	4M	16B	12h
Respawn [16]	SN-SN	2013	*Sensors:* – (T) ○ *Edge-L1:* 1-10k(S) ● *Edge-L1:* 1 ● *Cloud:* 1●	1M writes 1k reads	–	1 year (writes)
TorqueDB [40]	SN-SN	2020	*Edge-L1:* 12 ● *Edge-L2:* 3 ● *Cloud:* 1 ●	–	–	–
Antelope [111]	SN-IN	2011	*Sensors:* 20 ● *Sensors:* 40(E) ●	–	–	–
TinyDB [69]	SN-IN	2005	*Sensors:* 4-200(T) ●	600-200k	–	3 months
Presto [63]	SN-P	2009	*Sensors:* 20 ● *Sensors:* 20-100(T) ● *Edge-L1:* 1 ●	300-77k queries	–	1h - 5 days
EnviroStore [68]	SN-I	2007	*Sensors:* 8-36 (S) ●	–	–	1h
Vasilescu et al. [113]	SN-I	2005	*Sensors:* 3-8 ●	4k samples	27B	7 days
Search NDNoT [98]	SN-SE	2017	*Sensors:* 2 ○ *Edge-L1:* 2 ● *Servers:* 1 ●	2-10 searches	–	4-110s
IoT-SVK [31, 32]	SN-SE	2012	*Sensors:* 10k(T) ○ *Sensors:* 340k(G) ○ *Edge-L1:* – ● *Servers:* – ●	Single searches	–	–
Dyser [83]	SN-SE	2010	*Sensors:* 385(T) ● *Servers: 1* ●	–	–	3 months
Snoogle [117]	SN-SE	2008	*Sensors:* 1-30 ● *Edge-L1:* 1-8 ● *Edge-L2:* 0-1 ●	40-300 searches	4B	–
Meiklejohn et al. [72]	PoP	2018	*Edge-L1:* – ●	–	–	–
SEQ/Hybrid [67]	PoP	2007	*Edge-L1:* 5-10 ●	–	–	–
DBProxy [5]	PoP	2003	*Clients:* 1 ○ *Edge-L1:* 1 ● *Cloud:* 1 ●	–	–	1000s
3D-MADS [49]	AV	2019	*Cars:* 28590 taxis(T) ● *Cars:* 2 Robots ● *Edge-L1:* 40 RSU(S) ●	2k-2.5k transmissions	1024 B	–

Boehm et al. [12]	AV	2015	*Edge-L1:* 1 ● *Servers:* 1 ●	1	40 MB	0.268-95.8s
FogStore [46]	Other	2018	*Edge-L1:* 8 ●	–	–	–
SmartLite [65]	Other	2023	*Edge-L1:* 1 ●	–	–	–

Symbols: (T) Trace Replay (G) Generated – Not Available (S) Simulated (E) Emulated ● Stores data ○ Does not store data

with a significant number of operations that allow the evaluation of their scalability. In sum, more accurate simulations, and emulations of bigger scale should be explored for Sensor Networks. On trace replay alternatives, datasets that have the scale of billions of points per day would be useful to replicate the most demanding industrial scenarios.

Points of Presence (PoPs) consider clusters of up to 10 nodes. Future work could focus on increasing the number of replicas at different locations while minimizing the overhead incurred, in order to further improve latency, e.g., allowing a replica in each country a given service is offered.

Autonomous Vehicle (AV) systems consider traces from more than 28 000 vehicles. However, due to the critical and complex nature of the application domain, experiments with real vehicles carry considerable more value. From the works considered, the evaluation which is closest to a real scenario is that of Ho et al. [49] which considers a scenario with 2 robots.

Testing standardization is needed for all application domains, as each uses different timeframes and number and size of operations, preventing the comparison of the results between solutions targeting the same use case. Evaluations of **bigger scale** are also needed.

6 RELATED SURVEYS

Related surveys vary in degree of specificity. Some define edge, fog and related paradigms, and build taxonomies that cover all types of systems associated to them [50, 126]. Others cover a wide spectrum of data management systems, from DBMSs and Data Stores, to File Systems, and systems focusing on: security, analytics, forecasting, amongst others [76, 93]. With a more reduced scope, some focus on DBMSs specific to a particular application domain, such as sensor network DBMSs [29]. This survey sits in the middle of the latter two, presenting a holistic view of DBMSs in edge/fog scenarios.

Diallo et al. [29] survey sensor network data systems, dividing them in two approaches: the warehousing approach, equivalent to the SN-SN application domain; and the distributed approach, that fits into the previously defined SN-IN processing model.

Moysiadis et al. [76] analyzes distributed storage systems in *Fog computing*, and also the Social Internet of Things, and its storage needs. It is the closest to the work here presented, but it does not focus on databases/data stores in particular. They analyze general storage systems (e.g. file-systems), simulation tools, Storage as a Service, and various architectures (e.g., P2P, hybrid cloud/edge, ...), as well as present an overview of relevant research projects. Linking with this paper, the authors explore some publications targeting data distribution, dissemination, and replication in fog and edge environments. In comparison, our work considers a greater number of database solutions, and offers various database specific classification axes.

Sadri et al. [93] surveys data management in *Fog Computing*. It presents a taxonomy that divides publications into 3 main categories: Data Processing, Data Storage, and Data Security. This work differs from the one here presented as it does not focus on databases/data stores, and it focuses specifically on systems which use the term *"Fog"*, whereas many databases developed specifically for the edge/fog are associated to the term *"Edge"*. On privacy, their work supports the same conclusions

reached in this work, in that only a few papers mention privacy through data placement, while most look at encryption, authentication, and privacy-preserving aggregation.

Yousefpour et al. [126] present a high-level view of edge, fog, and related paradigms. Definitions for *Edge Computing*, *Fog Computing*, *Multi-Access Edge Computing*, and related paradigms are discussed. A topology of research topics is presented, e.g., scheduling, offloading, hardware, testbeds, architectures, amongst others. For each publication, the survey points out its main focus points, e.g. cost, energy, QoS, and security. This work spans across multiple areas of research.

Hong et al. [50], similarly to Yousefpour et al., follow a broader approach to surveying *Edge* and *Fog Computing*, looking at topics that span from physical devices, to service placement, node discovery, and others, providing a visual map of the identified areas. They do not explore in depth any particular theme, but rather present a few pertinent publications from each area. Relevant sections to the work here presented include the sections on "Replication", "Database cloning", and "Application-specific Data Replication".

Li et al. [62] also provide a wide view of edge and fog computing systems, from the perspectives of: concept definitions, architectures, system use cases and optimization objectives. Relevant works analyzed that are of value to this survey include Data Caching and Data replication systems.

7 FINAL REMARKS AND FUTURE RESEARCH DIRECTIONS

The edge enables new use cases and provides means for better system performance, providing services a competitive advantage in scenarios: **with limited connectivity, and energy; where privacy restrictions are in place; and when low latency is needed**. The edge is composed by a set of application domains that differ from the ones of cloud environments. In total, three domains have been identified, namely: Points of Presence, Sensor Networks, and Autonomous Vehicles.

Although edge and fog computing are considered recent topics, the first application of edge resources actually precedes public cloud services. CDNs were, and are used to serve web content with lower latency. They are based on Points of Presence, i.e., geographically distributed server-grade machines, that are situated closer to end-users. Database systems targeting PoPs focus on the development of geo-replication protocols that maximize the latency advantage provided by their geographical placement.

- Research on PoPs should focus on expanding the number of geographically-dispersed replicas supported to further explore latency benefits, while minimizing the overhead incurred by the coordination effort.

Sensor Networks dominate research in data storage at the edge, especially Time-Series databases, spanning more than half of surveyed works. The focus is on providing Sink-Node databases capable of ingesting data streams from many parallel sensors, and able to fit in resource constrained nodes.

- Research on Sensor Networks should explore hybrid Sink-Node and In-Network systems, combining optimizations at both the sensor and sink node level.
- Research on Sensor Networks should combine push and pull-based approaches to accommodate live metric monitoring while saving energy on data which is only requested sporadically.

In Predictive Sensor Networks, recent advances in Machine Learning techniques and hardware for edge nodes allow for more complex models to move to the edge.

- Research on Sensor Networks should take into account Machine Learning advances for the edge, expanding predictive based approaches to more complex data patterns.

Sensor Network systems also explore a separate research axis where Search-Engines are used to efficiently locate sensors with a given desired state. Although such systems have been tested with a

significant amount of traced sensors, the amount of queries posed to the system during evaluations is small and does not explore scalability in a concurrent client scenario.

- Future work in Search-Engine Sensor Networks should consider bigger workloads, and analyze system scalability when under stress by multiple concurrent clients.

On Autonomous Vehicles, the focus is on data transmission, efficient representation and compression of 3D map data. Systems are often inserted into multi-layer architectures which encompass a central point where all map data is kept, and multiple edge nodes, or Roadside Units, which contain partial replicas of data to provide passing vehicles information about upcoming stretches of road. Ensuring consistency of map updates between different nodes within defined time constraints is of utmost importance in scenarios where such information can lead to collision avoidance.

- Future work must focus on efficient Client Communication Managers and on Query Engines for 3D map data processing and storage, and in Replication Services with time interval requirements.
- Future work should focus on further exploring moving object representations, minimizing updates and maintaining an acceptable degree of position uncertainty.
- Future work should consider real-life evaluation of systems, as vehicle's complex nature makes it hard to accurately represent these systems under simulation scenarios.

Latency is often used as an argument for placing computation and storage at the edge. However, the **difference between *Cloud* and *Edge* deployments is sometimes only of a few milliseconds.** Furthermore, the latency gain between *Edge-enabled* and *Cloud-only* deployments is not analyzed for most publications. Looking at latency requirements, users expect increasingly lower latency values, and research shows that **a difference of 250ms or more between competing web services and search-engines can provide an advantage** in user adoption. For **interactive systems**, such as map panning applications, expected latency value can be as low as **100ms**. Finally, for **Autonomous Vehicles, 100ms** is the expected reaction time.

- Research should compare *Cloud*, *Edge* and *Fog* deployment latency to measure the latency gain incurred for data at the edge, especially in systems using latency as their main driver.
- Research should establish a common methodology for latency evaluation, to enable the direct comparison of competing solutions' latencies.
- Research should focus on quantifying latency requirements in critical systems, and for competitive advantage purposes, to understand the level of latency gain needed to provide a meaningful improvement.

The hardware being used for experiments at the edge **is not state-of-the-art**. Publications often consider hardware devices for their popularity. In some cases, cloud infrastructures are being used to play the role of edge nodes.

- Research should ensure the hardware chosen for nodes in layers in the *Edge-Cloud* continuum match those in production scenarios, and to the most adequate hardware available.

Edge nodes are said to be energy limited, however **energy limitations are not well understood**. While a battery powered node needs to prolong lifetime, many nodes such as Gateways, PoPs, Roaside Units, Sink Nodes, and others are usually powered by mains power in surveyed publications, without compelling proof that the case will be the same in real-life deployments, as they may be limited by energy harvesting scenarios. **Not all sensor network oriented use cases take into account the energy limited nature of sensors**, namely the SN-SN and SN-SE use cases.

- Research should expand energy consumption concerns to other use cases, such as in SN-SN and SN-SE systems. Energy limitations of other types of devices, e.g., PoPs, Gateways and Sink Nodes should be evaluated, and specific energy optimization mechanisms be developed.

Overall, a clear **standard in what refers to system evaluation is lacking**. Most works use workloads which are not realistic, and consider few nodes at each layer. There are **no suitable benchmarking solutions** for the presented application domains, which leads evaluations to differ considerably, even amongst publications targeting the same application domain. **No publication surveyed compared itself against competing solutions**. In Sensor Networks, node simulations and emulation tools are used due to the high number of nodes involved, but those simulations often only accurately simulate certain aspects of a node, e.g., its network communication component.

- Research should develop relevant benchmarks focusing on database performance evaluation for the application domains presented. The availability of more workload traces would also benefit system evaluation.
- Research should target the development of specialized simulation tools that enable node simulations that are both accurate and able to scale to significant sizes.
- Research should compare newly developed systems against already existing systems targeting the same application domain.

Other technologies could be analyzed as to their applicability to edge-enabled database systems, e.g., delegating databases tasks, such as transaction execution, to network devices, a technique currently being explored on cloud infrastructure [53]; and the use of persistent memories on edge devices, as faster mediums for write operations requiring persistence [99, 123].

REFERENCES

[1] Alex Afanasyev, Jeff Burke, Tamer Refaei, Lan Wang, Beichuan Zhang, and Lixia Zhang. 2018. A brief introduction to named data networking. In *MILCOM 2018-2018 IEEE Military Communications Conf. (MILCOM)*. IEEE, New York, NY, 1–6.

[2] Lorenzo Affetti. 2017. Consistent Stream Processing: Doctoral Symposium. In *Proc. of the 11th ACM Int. Conf. on Distributed and Event-based Systems*. ACM, New York, NY, USA, 355–358.

[3] Akamai. 2024. Company history. Retrieved February 06, 2024 from https://www.akamai.com/company/company-history

[4] Siavash Alamouti, Fay Arjomandi, and Michel Burger. 2022. Hybrid Edge Cloud: A Pragmatic Approach for Decentralized Cloud Computing. *IEEE Communications Magazine* 60, 9 (2022), 16–29.

[5] Khalil Amiri, Sanghyun Park, Renu Tewari, and Sriram Padmanabhan. 2003. DBProxy: A Dynamic Data Cache for Web Applications. In *Proc. 19th Int. Conf. on Data Engineering*. IEEE, New York, NY, 821–821.

[6] Ioannis Arapakis, Xiao Bai, and B. Cambazoglu. 2014. Impact of Response Latency on User Behavior in Web Search. In *Proc. of the 37th Int. ACM SIGIR Conf. on Research & Development in Information Retrieval*. ACM, New York, NY, USA, 103–112.

[7] AWS. 2008. What is Amazon CloudFront? Retrieved January 30, 2023 from https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Introduction.html

[8] AWS. 2022. Amazon CloudFront. Retrieved October 04, 2022 from https://aws.amazon.com/pt/cloudfront/

[9] Azure. 2022. Azure Content Delivery Network. Retrieved October 04, 2022 from https://azure.microsoft.com/en-gb/products/cdn/#overview

[10] Kaustubh Beedkar, David Brekardin, Jorge-Anulfo Quiané-Ruiz, and Volker Markl. 2021. Compliant Geo-Distributed Data Processing in Action. *Proc. VLDB Endow.* 14, 12 (2021), 2843–2846.

[11] Burton Bloom. 1970. Space/Time Trade-Offs in Hash Coding with Allowable Errors. *Commun. ACM* 13, 7 (1970), 422–426.

[12] J Boehm and K Liu. 2015. NoSQL for Storage and Retrieval of Large LiDAR Data Collections. *Int. Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences (ISPRS)* XL-3/W3 (2015), 577–582.

[13] Andrea Bonci, Massimiliano Pirani, and Sauro Longhi. 2017. An Embedded Database Technology Perspective in Cyber-Physical Production Systems. *Procedia Manufacturing* 11 (2017), 830–837.

[14] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog Computing and its Role in the Internet of Things. In *Proc. of the First Edition of the MCC Workshop on Mobile Cloud Computing*. ACM, New York, NY, USA, 13–16.

[15] Jake D. Brutlag, Hilary Hutchinson, and Maria Stone. 2008. User Preference and Search Engine Latency. In *JSM Proceedings, Quality and Productivity Research Section*. Alexandria, VA.

[16] Maxim Buevich, Anne Wright, Randy Sargent, and Anthony Rowe. 2013. Respawn: A Distributed Multi-Resolution Time-Series Datastore. In *IEEE 34th Real-Time Systems Symp.* IEEE, New York, NY, 288–297.

[17] Yu Cao, Songqing Chen, Peng Hou, and Donald Brown. 2015. FAST: A Fog Computing Assisted Distributed Analytics System to Monitor Fall for Stroke Mitigation. In *IEEE Int. Conf. on Networking, Architecture and Storage (NAS)*. IEEE, New York, NY, 2–11.

[18] Nicholas Carr. 2000. On the Edge: An Interview with Akamai's George Conrades. Retrieved August 19, 2022 from https://hbr.org/2000/05/on-the-edge-an-interview-with-akamais-george-conrades

[19] Batyr Charyyev, Engin Arslan, and Mehmet Hadi Gunes. 2020. Latency Comparison of Cloud Datacenters and Edge Servers. In *IEEE Global Communications Conf. (GLOBECOM)*. IEEE, New York, NY, 1–6.

[20] Zhuo Chen, Wenlu Hu, Junjue Wang, Siyan Zhao, Brandon Amos, Guanhang Wu, Kiryong Ha, Khalid Elgazzar, Padmanabhan Pillai, Roberta Klatzky, et al. 2017. An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance. In *Proc. of the Second ACM/IEEE Symp. on Edge Computing*. ACM/IEEE, New York, NY, 1–14.

[21] Giulia Cisotto, Edoardo Casarin, and Stefano Tomasin. 2020. Requirements and Enablers of Advanced Healthcare Services over Future Cellular Systems. *IEEE Communications Magazine* 58, 3 (mar 2020), 76–81.

[22] Google Cloud. 2022. Cloud CDN. Retrieved October 04, 2022 from https://cloud.google.com/cdn

[23] OpenFog Consortium et al. 2018. IEEE standard for adoption of OpenFog reference architecture for fog computing. *IEEE Std 1934-2018* (2018), 1–176.

[24] Brian Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proc. of the 1st ACM Symp. on Cloud computing*. ACM, New York, NY, USA, 143–154.

[25] Madeleine IG Daepp, Alex Cabral, Vaishnavi Ranganathan, Vikram Iyer, Scott Counts, Paul Johns, Asta Roseway, Charlie Catlett, Gavin Jancke, Darren Gehring, et al. 2022. Eclipse: an end-to-end platform for low-cost, hyperlocal environmental sensing in cities. In *2022 21st ACM/IEEE Int. Conf. on Information Processing in Sensor Networks (IPSN)*. ACM/IEEE, New York, NY, 28–40.

[26] Alex Davies. 2014. Cisco pushes IoT analytics to the extreme edge with mist computing. Retrieved February 6, 2024 from https://rethinkresearch.biz/articles/cisco-pushes-iot-analytics-extreme-edge-mist-computing-2/

[27] Alan Demers, Johannes Gehrke, Rajmohan Rajaraman, Niki Trigoni, and Yong Yao. 2003. The Cougar Project: a Work-In-Progress Report. *ACM Sigmod Record* 32, 4 (2003), 53–59.

[28] Sergio Di Martino, Luca Fiadone, Adriano Peron, Alberto Riccabone, and Vincenzo Vitale. 2019. Industrial Internet of Things: Persistence for Time Series with NoSQL Databases. In *IEEE 28th Int. Conf. on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE, New York, NY, 340–345.

[29] Ousmane Diallo, Joel JPC Rodrigues, Mbaye Sene, and Jaime Lloret. 2013. Distributed Database Management Techniques for Wireless Sensor Networks. *IEEE Transactions on Parallel and Distributed Systems* 26, 2 (2013), 604–620.

[30] John Dilley, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, and Bill Weihl. 2002. Globally Distributed Content Delivery. *IEEE Internet Computing* 6, 5 (2002), 50–58.

[31] Zhiming Ding, Zhikui Chen, and Qi Yang. 2014. IoT-SVKSearch: a real-time multimodal search engine mechanism for the internet of things. *Int. Journal of Communication Systems* 27, 6 (2014), 871–897.

[32] Zhiming Ding, Xu Gao, Limin Guo, and Qi Yang. 2012. A Hybrid Search Engine Framework for the Internet of Things Based on Spatial-Temporal, Value-Based, and Keyword-Based Conditions. In *IEEE Int. Conf. on Green Computing and Communications*. IEEE, New York, NY, 17–25.

[33] Min Dong, Haozhao Zhong, Boyu Sun, Sheng Bi, and Yi Cai. 2021. SardineDB: A Distributed Database on the Edge of the Network. In *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Int. Conf. on Web and Big Data*. Springer, Cham, 186–193.

[34] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. 2021. Rocksdb: Evolution of development priorities in a key-value store serving large-scale applications. *ACM Transactions on Storage (TOS)* 17, 4 (2021), 1–32.

[35] Schahram Dustdar, Cosmin Avasalcai, and Ilir Murturi. 2019. Edge and Fog Computing: Vision and Research Challenges. In *IEEE Int. Conf. on Service-Oriented System Engineering (SOSE)*. IEEE, New York, NY, 96–9609.

[36] Joakim Eriksson, Adam Dunkels, Niclas Finne, Fredrik Osterlind, and Thiemo Voigt. 2007. MSPsim - an extensible simulator for msp430-equipped sensor boards. In *Proc. of the European Conf. on Wireless Sensor Networks (EWSN), Poster/Demo session*, Vol. 118. Springer, Cham.

[37] ETSI. 2024. Standards. Retrieved February 6, 2024 from https://www.etsi.org/standards-search#page=1&version=1&onApproval=1&published=1&sort=3&TB=826,,835,,874

[38] Autoware Foundation. 2019. Autoware - The World's Leading Open-Source Software Project for Autonomous Driving. Retrieved November 29, 2022 from https://github.com/autowarefoundation/autoware

[39] OPC Foundation. 2024. OpenFog. Retrieved February 6, 2024 from https://opcfoundation.org/markets-collaboration/openfog/

[40] Dhruv Garg, Prathik Shirolkar, Anshu Shukla, and Yogesh Simmhan. 2020. Torquedb: Distributed querying of time-series data from edge-local storage. In *Euro-Par 2020: Parallel Processing: 26th Int. Conf. on Parallel and Distributed Computing, Warsaw, Poland, August 24–28, 2020, Proc. 26*. Springer, Cham, 281–295.
[41] Google. 2022. Edge Computing Google Trend. Retrieved November 15, 2022 from https://trends.google.pt/trends/explore?date=all&q=Edge%20Computing
[42] Ulrich Greveler, Peter Glösekötterz, Benjamin Justusy, and Dennis Loehr. 2012. Multimedia Content Identification Through Smart Meter Power Usage Profiles. In *Proc. of the Int. Conf. on Information and Knowledge Engineering (IKE)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 1.
[43] Piotr Grzesik and Dariusz Mrozek. 2020. Comparative Analysis of Time Series Databases in the Context of Edge Computing for Low Power Sensor Networks. In *Int. Conf. on Computational Science (ICCS)*. Springer, Cham, 371–383.
[44] Yu Gu and Tian He. 2010. Bounding communication delay in energy harvesting sensor networks. In *2010 IEEE 30th Int. Conf. on Distributed Computing Systems*. IEEE, New York, NY, 837–847.
[45] Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, Frank Liu, et al. 2022. Manu: a cloud native vector database management system. *Proc. of the VLDB Endowment* 15, 12 (2022), 3548–3561.
[46] Harshit Gupta and Umakishore Ramachandran. 2018. Fogstore: A Geo-Distributed Key-Value Store Guaranteeing Low Latency for Strongly Consistent Access. In *Proc. of the 12th ACM Int. Conf. on Distributed and Event-based Systems*. ACM, New York, NY, USA, 148–159.
[47] Susan Hall. 2020. IoTDB Provides Data Management for Industrial Edge IT. Retrieved July 14, 2022 from https://thenewstack.io/iotdb-provides-data-management-for-industrial-edge-it/
[48] Joseph Hellerstein, Michael Stonebraker, James Hamilton, et al. 2007. Architecture of a Database System. *Foundations and Trends in Databases* 1, 2 (2007), 141–259.
[49] Ivan Ho, Sid Chau, Elmer Magsino, and Kanghao Jia. 2019. Efficient 3D Road Map Data Exchange for Intelligent Vehicles in Vehicular Fog Networks. *IEEE Transactions on Vehicular Technology* 69, 3 (2019), 3151–3165.
[50] Cheol-Ho Hong and Blesson Varghese. 2019. Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms. *ACM Computing Surveys (CSUR)* 52, 5 (2019), 1–37.
[51] ASPRS The Imaging and Geospatial Information Society. 2013. LAS SPECIFICATION. Retrieved February 20, 2023 from https://www.asprs.org/wp-content/uploads/2010/12/LAS_1_4_r13.pdf
[52] influxdata. 2013. It's About Time. Build on InfluxDB. Retrieved December 26, 2022 from https://www.influxdata.com/
[53] Matthias Jasny, Lasse Thostrup, Tobias Ziegler, and Carsten Binnig. 2022. P4DB - The Case for In-Network Oltp. In *Proc. of the Int. Conf. on Management of Data*. ACM, New York, NY, USA, 1375–1389.
[54] Søren Kejser Jensen, Christian Thomsen, and Torben Bach Pedersen. 2023. *ModelarDB: integrated model-based management of time series from edge to cloud*. Springer, Berlin, Heidelberg, 1–33.
[55] Guodong Jin, Xiyang Feng, Ziyi Chen, Chang Liu, and Semih Salihoglu. 2023. KÙZU Graph Database Management System. In *13th Conf. on Innovative Data Systems Research (CIDR)*.
[56] Fiodar Kazhamiaka, Matei Zaharia, and Peter Bailis. 2021. Challenges and Opportunities for Autonomous Vehicle Query Systems.. In *Conf. on Innovative Data Systems Research (CIDR)*.
[57] Personal Information Protection Comittee (South Korea). 2011. Protection of Personal Information Act. Retrieved January 30, 2023 from https://www.privacy.go.kr/eng/laws_view.do?nttId=8186&imgNo=33
[58] Timothy Krentz, Abhishek Dubey, and Gabor Karsai. 2019. Towards An Edge-Located Time-Series Database. In *IEEE 22nd Int. Symp. on Real-Time Distributed Computing (ISORC)*. IEEE, New York, NY, 151–154.
[59] Leslie Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* 21, 7 (1978), 558–565.
[60] Chien-I Lee, Meng-Yao Lin, Chia-Lin Yang, and Yen-Kuang Chen. 2019. IoTBench: A Benchmark Suite for Intelligent Internet of Things Edge Devices. In *2019 IEEE Int. Conf. on Image Processing (ICIP)*. IEEE, New York, NY, 170–174.
[61] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. 2003. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *Proc. of the 1st Int. Conf. on Embedded Networked Sensor Systems*. ACM, New York, NY, USA, 126–137.
[62] Chao Li, Yushu Xue, Jing Wang, Weigong Zhang, and Tao Li. 2018. Edge-oriented computing paradigms: A survey on architecture design and system management. *ACM Computing Surveys (CSUR)* 51, 2 (2018), 1–34.
[63] Ming Li, Deepak Ganesan, and Prashant Shenoy. 2009. PRESTO: Feedback-Driven Data Management in Sensor Networks. *IEEE/ACM Transactions on Networking* 17, 4 (2009), 1256–1269.
[64] Wenbin Li and Matthieu Liewig. 2020. A Survey of AI Accelerators for Edge Environment. In *Trends and Innovations in Information Systems and Technologies*. Springer, Cham, 35–44.
[65] Qiuru Lin, Sai Wu, Junbo Zhao, Jian Dai, Meng Shi, Gang Chen, and Feifei Li. 2023. SmartLite: A DBMS-Based Serving System for DNN Inference in Resource-Constrained Environments. *Proc. of the VLDB Endow.* 17, 3 (2023), 278–291.

[66] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md. Haque, Lingjia Tang, and Jason Mars. 2018. The Architectural Implications of Autonomous Driving: Constraints and Acceleration. In *Proc. of the Twenty-Third Int. Conf. on Architectural Support for Programming Languages and Operating Systems*. ACM, New York, NY, USA, 751–766.

[67] Yi Lin, Bettina Kemme, Marta Patino-Martinez, and Ricardo Jimenez-Peris. 2007. Enhancing Edge Computing with Database Replication. In *26th IEEE Int. Symp on Reliable Distributed Systems (SRDS 2007)*. IEEE, New York, NY, 45–54.

[68] Liqian Luo, Chengdu Huang, Tarek Abdelzaher, and John Stankovic. 2007. Envirostore: A Cooperative Storage System for Disconnected Operation in Sensor Networks. In *26th IEEE Int. Conf. on Computer Communications (INFOCOM)*. IEEE, New York, NY, 1802–1810.

[69] Samuel R Madden, Michael J Franklin, Joseph M Hellerstein, and Wei Hong. 2005. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *ACM Transactions on Database Systems (TODS)* 30, 1 (2005), 122–173.

[70] Yaser Mansouri, Victor Prokhorenko, Faheem Ullah, and Ali Babar. 2021. Evaluation of Distributed Databases in Hybrid Clouds and Edge Computing: Energy, Bandwidth, and Storage Consumption. (2021). https://doi.org/10.48550/arXiv.2109.07260 arXiv:arXiv/2109.07260

[71] Jonathan McChesney, Nan Wang, Ashish Tanwer, Eyal de Lara, and Blesson Varghese. 2019. Defog: Fog Computing Benchmarks. In *Proc. of the 4th ACM/IEEE Symp. on Edge Computing*. ACM/IEEE, New York, NY, USA, 47–58.

[72] Christopher Meiklejohn, Heather Miller, and Zeeshan Lakhani. 2018. Towards a Solution to the Red Wedding Problem. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge)*. USENIX, Boston, MA.

[73] Dirk Merkel et al. 2014. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.* 239, 2 (2014), 2.

[74] Ron Miller. 2016. How AWS Came to Be. Retrieved July 13, 2022 from https://techcrunch.com/2016/07/02/andy-jassys-brief-history-of-the-genesis-of-aws/

[75] Sumit Kumar Monga, Sheshadri K Ramachandra, and Yogesh Simmhan. 2019. ElfStore: A resilient data storage service for federated edge and fog resources. In *2019 IEEE Int. Conf. on Web Services (ICWS)*. IEEE, New York, NY, 336–345.

[76] Vasileios Moysiadis, Panagiotis Sarigiannidis, and Ioannis Moscholios. 2018. Towards Distributed Data Management in Fog Computing. *Wireless Communications and Mobile Computing* 2018 (2018).

[77] Fiona Nah. 2004. A Study on Tolerable Waiting Time: How Long are Web Users Willing to Wait? *Behaviour & Information Technology* 23, 3 (2004), 153–163.

[78] Brian Noble, Mahadev Satyanarayanan, Dushyanth Narayanan, James Tilton, Jason Flinn, and Kevin Walker. 1997. Agile Application-Aware Adaptation for Mobility. *ACM SIGOPS Operating Systems Review* 31, 5 (1997), 276–287.

[79] Federal Ministry Republic of Austria. 2018. DSG. Retrieved January 30, 2023 from https://www.bmf.gv.at/en/data-protection.html

[80] Federal Republic of Brazil. 2018. Lei Geral de Proteção de Dados Pessoais (LGPD). Retrieved January 30, 2023 from https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm

[81] Republic of South Africa. 2013. Protection of Personal Information Act. Retrieved January 30, 2023 from https://www.gov.za/sites/default/files/gcis_document/201409/3706726-11act4of2013protectionofpersonalinforcorrect.pdf

[82] Official Journal of the European Union. 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council. Retrieved January 30, 2023 from https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN#d1e4227-1-1

[83] Benedikt Ostermaier, Kay Römer, Friedemann Mattern, Michael Fahrmair, and Wolfgang Kellerer. 2010. A Real-Time Search Engine for the Web of Things. In *Internet of Things (IOT)*. IEEE, New York, NY, 1–8.

[84] James Otto, Mohammad Najdawi, and Karen Caron. 2000. Web-User Satisfaction: An Exploratory Study. *Journal of Organizational and End User Computing (JOEUC)* 12, 4 (2000), 3–10.

[85] John Paparrizos, Chunwei Liu, Bruno Barbarioli, Johnny Hwang, Ikraduya Edian, Aaron J Elmore, Michael J Franklin, and Sanjay Krishnan. 2021. VergeDB: A Database for IoT Analytics on Edge Devices. In *Conf. on Innovative Data Systems Research (CIDR)*.

[86] Imtiaz Parvez, Ali Rahmati, Ismail Guvenc, Arif Sarwat, and Huaiyu Dai. 2018. A Survey on Low Latency Towards 5G: RAN, Core Network and Caching Solutions. *IEEE Communications Surveys & Tutorials* 20, 4 (2018), 3098–3130.

[87] Tobias Pfandzelter and David Bermbach. 2021. Towards Predictive Replica Placement for Distributed Data Stores in Fog Environments. In *IEEE Int. Conf. on Cloud Engineering (IC2E)*. IEEE, New York, NY, 280–281.

[88] Wolf-Bastian Pöttner, Hans Seidel, James Brown, Utz Roedig, and Lars Wolf. 2014. Constructing Schedules for Time-Critical Data Delivery in Wireless Sensor Networks. *ACM Transactions on Sensor Networks (TOSN)* 10, 3 (2014), 1–31.

[89] J. Queralta, Tuan Gia, Hannu Tenhunen, and Tomi Westerlund. 2019. Edge-AI in LoRa-based Health Monitoring: Fall Detection System with Fog Computing and LSTM Recurrent Neural Networks. In *42nd Int. Conf. on Telecommunications and Signal Processing (TSP)*. IEEE, New York, NY, 601–604.

[90] Amir Rahmani, Tuan Gia, Behailu Negash, Arman Anzanpour, Iman Azimi, Mingzhe Jiang, and Pasi Liljeberg. 2018. Exploiting Smart E-Health Gateways at the Edge of Healthcare Internet-of-Things: A Fog Computing Approach.

Future Generation Computer Systems 78 (2018), 641–658.

[91] Sylvia Ratnasamy, Brad Karp, Li Yin, Fang Yu, Deborah Estrin, Ramesh Govindan, and Scott Shenker. 2002. GHT: A Geographic Hash Table for Data-Centric Storage. In *Proc. of the 1st ACM Int. Workshop on Wireless Sensor Networks and Applications*. ACM, New York, NY, USA, 78–87.

[92] David Reinsel, John Gantz, and John Rydning. 2017. *Data Age 2025: The Evolution of Data to Life-Critical. Don't Focus on Big Data; Focus on Data That's Big*. Technical Report. Int. Data Corporation (IDC).

[93] Ali Sadri, Amir Rahmani, Morteza Saberikamarposhti, and Mehdi Hosseinzadeh. 2021. Fog Data Management: A Vision, Challenges, and Future Directions. *Journal of Network and Computer Applications* 174 (2021), 102882.

[94] Victor Sarker, J. Queralta, Tuan Gia, Hannu Tenhunen, and Tomi Westerlund. 2019. A Survey on LoRa for IoT: Integrating Edge Computing. In *Fourth Int. Conf. on Fog and Mobile Edge Computing (FMEC)*. IEEE, New York, NY, 295–300.

[95] Mahadev Satyanarayanan. 2001. Pervasive Computing: Vision and Challenges. *IEEE Personal communications* 8, 4 (2001), 10–17.

[96] Mahadev Satyanarayanan. 2017. The Emergence of Edge Computing. *Computer* 50, 1 (2017), 30–39. https://doi.org/10.1109/MC.2017.9

[97] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. 2009. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE pervasive Computing* 8, 4 (2009), 14–23.

[98] Divya Saxena, Vaskar Raychoudhury, and Christian Becker. 2017. An NDNoT Based Efficient Object Searching Scheme for Smart Home Using RFIDs. In *Proc. of the 18th Int. Conf. on Distributed Computing and Networking*. ACM, New York, NY, USA, 1–6.

[99] Anil Shanbhag, Nesime Tatbul, David Cohen, and Samuel Madden. 2020. Large-Scale In-Memory Analytics on Intel Optane DC Persistent Memory. In *Proc. of the 16th Int. Workshop on Data Management on New Hardware*. ACM, New York, NY, USA, 1–8.

[100] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-Free Replicated Data Types. In *Symp. on Self-Stabilizing Systems*. Springer, Berlin, Heidelberg, 386–400.

[101] Scott Shenker, Sylvia Ratnasamy, Brad Karp, Ramesh Govindan, and Deborah Estrin. 2003. Data-Centric Storage in Sensornets. *ACM SIGCOMM Computer Communication Review* 33, 1 (2003), 137–142.

[102] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.

[103] Meryem Simsek, Adnan Aijaz, Mischa Dohler, Joachim Sachs, and Gerhard Fettweis. 2016. 5G-Enabled Tactile Internet. *IEEE Journal on Selected Areas in Communications* 34, 3 (2016), 460–473.

[104] Samriddhi Singla, Ahmed Eldawy, Tina Diao, Ayan Mukhopadhyay, and Elia Scudiero. 2021. Experimental study of big raster and vector database systems. In *2021 IEEE 37th Int. Conf. on Data Engineering (ICDE)*. IEEE, New York, NY, 2243–2248.

[105] Karim Sonbol, Öznur Özkasap, Ibrahim Al-Oqily, and Moayad Aloqaily. 2020. EdgeKV: Decentralized, Scalable, and Consistent Storage for the Edge. *J. Parallel and Distrib. Comput.* 144 (2020), 28–40.

[106] Statista. 2022. Share of Corporate Data Stored in the Cloud in Organizations Worldwide From 2015 to 2022. Retrieved January 9, 2023 from https://www.statista.com/statistics/1062879/worldwide-cloud-storage-of-corporate-data/

[107] Retail Info Systems. 2008. Retail Web Site Performance – Consumer Reaction to a Poor Online Shopping Experience. Retrieved September 23, 2022 from https://risnews.com/retail-web-site-performance-consumer-reaction-poor-online-shopping-experience

[108] Nihit Tandon. 2018. Netflix Content Distribution Through Open Connect. Retrieved October 04, 2022 from https://blog.apnic.net/2018/06/20/netflix-content-distribution-through-open-connect/

[109] TPC. 2000. TPC-W. Retrieved January 11, 2023 from https://www.tpc.org/tpcw/

[110] TPC. 2017. TPCx-IoT Version 1 and Version 2. Retrieved September 27, 2022 from https://www.tpc.org/tpcx-iot/default5.asp

[111] Nicolas Tsiftes and Adam Dunkels. 2011. A Database in Every Sensor. In *Proc. of the 9th ACM Conf. on Embedded Networked Sensor Systems*. ACM, New York, NY, USA, 316–332.

[112] European Union. 2018. General Data Protection Regulation. Retrieved January 30, 2023 from https://gdpr.eu/tag/gdpr/

[113] Iuliu Vasilescu, Keith Kotay, Daniela Rus, Matthew Dunbabin, and Peter Corke. 2005. Data Collection, Storage, and Retrieval With an Underwater Sensor Network. In *Proc. of the 3rd Int. Conf. on Embedded Networked Sensor Systems*. ACM, New York, NY, USA, 154–165.

[114] Thiemo Voigt, Martin Bor, Utz Roedig, and Juan Alonso. 2016. Mitigating Inter-network Interference in LoRa Networks. In *Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks*. Junction Publishing, USA, 323–328.

[115] Chen Wang, Xiangdong Huang, Jialin Qiao, Tian Jiang, Lei Rui, Jinrui Zhang, Rong Kang, Julian Feinauer, Kevin McGrail, Peng Wang, et al. 2020. Apache IoTDB: Time-Series Database For Internet of Things. *Proc. VLDB Endow.* 13,

12 (2020), 2901–2904.

[116] Chen Wang, Jialin Qiao, Xiangdong Huang, Shaoxu Song, Haonan Hou, Tian Jiang, Lei Rui, Jianmin Wang, and Jiaguang Sun. 2023. Apache IoTDB: A Time Series Database for IoT Applications. *Proc. of the ACM on Management of Data* 1, 2 (2023), 1–27.

[117] Haodong Wang, Chiu Tan, and Qun Li. 2008. Snoogle: A search Engine for the Physical World. In *27th IEEE Int. Conf. on Computer Communications (INFOCOM)*. IEEE, New York, NY, 1382–1390.

[118] Xiaofei Wang, Yiwen Han, Victor Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. 2020. Convergence of Edge Computing and Deep Learning: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials* 22, 2 (2020), 869–904.

[119] Mark Weiser. 1991. The Computer for the 21 st Century. *Scientific american* 265, 3 (1991), 94–105.

[120] Philip Wette, Martin Dräxler, Arne Schwabe, Felix Wallaschek, Mohammad Zahraee, and Holger Karl. 2014. Maxinet: Distributed Emulation of Software-Defined Networks. In *IFIP Networking Conf.* IEEE, New York, NY, 1–9.

[121] Ouri Wolfson, Prasad Sistla, Bo Xu, Jutai Zhou, and Sam Chamberlain. 1999. DOMINO: Databases for Moving Objects Tracking. *ACM SIGMOD Record* 28, 2 (1999), 547–549.

[122] Fan Wu, Chunkai Qiu, Taiyang Wu, and Mehmet Yuce. 2021. Edge-Based Hybrid System Implementation for Long-Range Safety and Healthcare IoT Applications. *IEEE Internet of Things Journal* 8, 12 (2021), 9970–9980.

[123] Yinjun Wu, Kwanghyun Park, Rathijit Sen, Brian Kroth, and Jaeyoung Do. 2020. Lessons Learned from the Early Performance Evaluation of Intel Optane DC Persistent Memory in DBMS. In *Proc. of the 16th Int. Workshop on Data Management on New Hardware*. ACM, New York, NY, USA, 1–3.

[124] Yang Yang, Qiang Cao, and Hong Jiang. 2019. EdgeDB: An Efficient Time-Series Database for Edge Computing. *IEEE Access* 7 (2019), 142295–142307.

[125] Yong Yao and Johannes Gehrke. 2002. The Cougar Approach to In-Network Query Processing in Sensor Networks. *ACM Sigmod Record* 31, 3 (2002), 9–18.

[126] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason Jue. 2019. All One Needs to Know About Fog Computing and Related Edge Computing Paradigms: A Complete Survey. *Journal of Systems Architecture* 98 (2019), 289–330.

[127] Shen Yulong, Xi Ning, Pei Qingqi, Ma Jianfeng, Xu Qijian, and Wu Zuoshun. 2011. Distributed Storage Schemes for Controlling Data Availability in Wireless Sensor Networks. In *Seventh Int. Conf. on Computational Intelligence and Security*. IEEE, New York, NY, 545–549.

[128] Mary Zhang. 2022. How Much Does it Cost to Build a Data Center? Retrieved January 30, 2023 from https://dgtlinfra.com/how-much-does-it-cost-to-build-a-data-center/