International Journal of Scientific Research and Modern Technology (IJSRMT) ijsrmt.com
Volume 1, Issue 12, 2022
DOI: https://doi.org/10.38124/ijsrmt.v1i12.1187

Making IoT Data Processing Scalable: Architectures That Actually Work

Vamsee Pamisetty[1]

[1]Middleware Architect
ORCID ID: 0009-0001-1148-1714

Publication Date: 2022/12/10

Abstract

Scalable Data Processing Architectures for Internet of Things Applications (2022) investigates the trade-offs involved in different architectural paradigms for processing data generated by the Internet of Things, covering considerations of latency, scalability, reliability, and governance. The analysis demonstrates that no single architecture is suitable for all IoT data-processing workloads and identifies edge and fog computing approaches as the most realistic choice at present. A cloud-centric approach offers advantages in terms of scale and elasticity but suffers from higher latency and a lack of resilience to network failures. Recent developments in data ingestion and transport, storage strategies, and stream processing frameworks are also examined, highlighting protocols, standards, storage models, and fault-tolerance techniques conducive to IoT workloads.

The rapid growth of the Internet of Things presents immense opportunities and challenges for society. Layered architectures that help solve these challenges by distributing computing, storage, and control elements can be mapped to specific use cases; the choice of architecture depends on the workloads associated with the various subsystems. Recent work demonstrates the need for structured high-level data interchange during the data-ingestion and transport phase of IoT data-processing pipelines. However, the scalable nature of the cloud computing paradigm remains appealing for many IoT data-processing workloads running on cloud infrastructures, because of the ease of elasticity and resource management it provides.

Keywords: *Scalable IoT Data Processing, Internet of Things Architectures, Edge Computing Paradigms, Fog Computing Models, Cloud-Centric IoT Processing, Latency–Scalability Trade-Offs, IoT Reliability And Resilience, Network Failure Tolerance, IoT Data Ingestion Pipelines, Structured Data Interchange, Stream Processing Frameworks, IoT Storage Strategies, Fault-Tolerant IoT Systems, IoT Protocols and Standards, Layered IoT Architectures, Distributed Computing for IoT, Elastic Cloud Resources, Resource Management In IoT, Governance of IoT Data, Workload-Specific IoT Design.*

I. INTRODUCTION

Scientific and engineering communities, together with the industry, are exploring the Internet of Things (IoT), a general paradigm that offers an extremely flexible and broadly applicable set of services and application scenarios in which a wide variety of physically embedded smart active or passive objects offer their data, information knowledge, and capabilities for use. A truly scalable architecture is a requirement for an IoT solution to address devices potentially numbering into the billions that span a diversity of classes, contexts, and uses. Such scale involves several dimensions in data handling, including offering highly generic services and services configurable with very low latency, supporting high-throughput periodic ingestion, ingesting data with bit rates that vary greatly in time (with relations to the underlying sensing phenomena), elastic scalability of computational processing, and, of course, elastic growth or overall cost-latency-performance impact for the system. At the same time, deployment, management, operation, and evolution

must be possible by a wide set of non-expert users and providers.

Different sets of trade-offs and degrees of scalability are brought about by the architectural choices behind IoT solutions, whether based on edge, fog, cloud, or combinations thereof. Selection of suitable trade-offs requires a thorough understanding of the objectives to achieve with the solution and, consequently, the properties of the phenomena the solution must support. This understanding should also be applied in the context of private or consortium-owned systems with much smaller scope than the current mega cloud providers, since these owners naturally have a different relation with the platform users and must provide such services with much reduced costs. The first step, therefore, is the identification of the full set of supported operations, and an analysis of their characteristics. Use cases are constructed by a top-down exploration of the functionality supported by the system, in both end-user and provider perspectives.

- *Overview of the Study*

The objectives are to define aims, significance, and research questions; justify relevance to scalable IoT data processing. Scalable Data Processing Architectures for Internet of Things Applications (2022).

The main objective is to define and synthesize the core elements of a contribution on architectures that can scale to IoT data levels. The difference in requirements between processing data generated by embedded systems interacting with the physical world as opposed to data naturally available on the Internet arises several key challenges that are discussed in the literature and addressed accordingly. The particular aspect of the adaptive data workflow proposed by this work is specifically focused. Systems running into bandwidth, latency, CEO, uptime, interruption or other constraints, where actors across the data path can cooperatively favour or assist processing, require a different design than systems which enjoy sufficient resources.

Scalability issues of data ingestion, storage, stream processing, architecture choice, data transport protocols and, particularly, schema evolution are discussed and possible directions provided. Many questions remain open yet some further avenues of exploration become visible. Scalable Data Processing Architectures for Internet of Things Applications (2022).

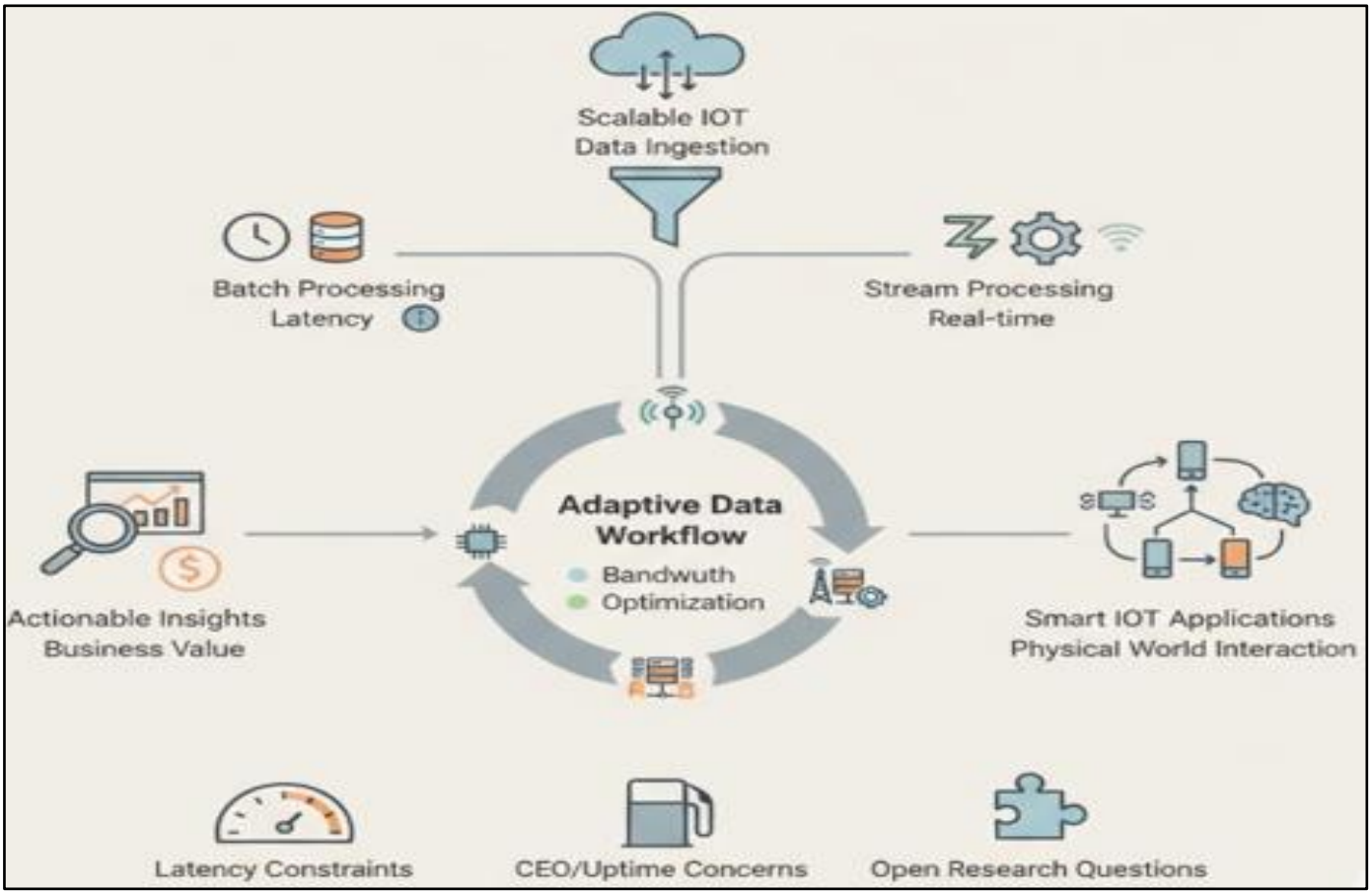

Fig 1 Adaptive Architectures for Resource-Constrained Environments: Navigating Scalability and Schema Evolution in IoT Data Workflows

II. BACKGROUND AND MOTIVATION

The widespread adoption of Internet of Things (IoT) technologies has created a significant shift from the Internet of People towards the Internet of Everything. The growing number of device connections and deployment of sensors in the physical world will enable previously unimagined services and applications, especially in smart cities. However, despite the great potential benefits, many IoT services and applications remain unproven or in pilot mode. The full-scale adoption of IoT technologies will depend more on analyzing and consolidating the accumulated data than on the continued development of technological solutions.

The IoT can facilitate predictive maintenance of equipment, improve decision making, prevent logistics problems, and avoid high-profile failures. Nevertheless, the communication and processing of the accumulated data remain problematic. Data generated by IoT devices present specific characteristics that challenge current architectures for distributing and processing data in cloud environments. Scalability and low latency remain crucial points of concern. Data access patterns introduce new requirements for storage technologies, and real-time analytics demand a new approach to data ingestion. The analysis of data processing and storage layers provides insights important for the evolution of current infrastructures and systems. Several architectural paradigms have been proposed to process and store this data. Each paradigm performs well for a subset of IoT applications; the remaining applications are usually ignored or addressed with ad hoc solutions. A generic analysis of the desired characteristics and operational constraints enables these paradigms to be compared in terms of trust, privacy, performance, management, and scalability. Such an analysis provides a broader view of the issues and trade-offs involved in data processing and storage, guides the choice of architecture according to the data and service envisaged, and indicates areas requiring further research.

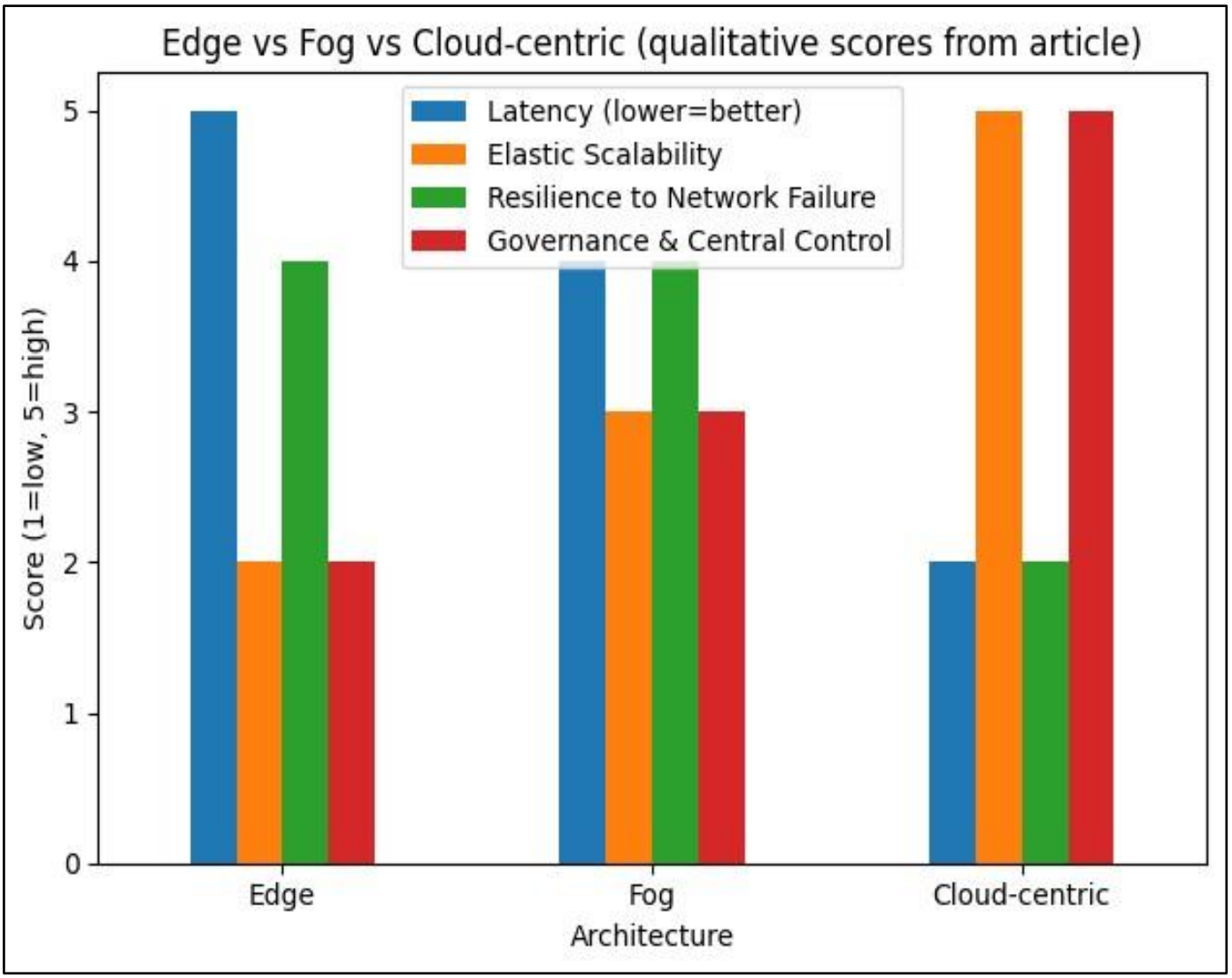

Fig 2 Architectural Latency–Scalability Trade-Off Model Across Edge, Fog, and Cloud Layers

- *Equation 1: End-to-End Latency Equations (Edge vs Fog vs Cloud)*

The repeatedly frames architecture choice as a latency vs scalability/reliability/governance trade-off.

➢ *Decompose End-to-End Latency*

For an IoT event generated at a device and producing an actionable output, model end-to-end latency as:

$$L_{e2e} = L_{sense} + L_{tx} + L_{queue} + L_{proc} + L_{store} + L_{return}$$

- *Step-by-Step Meaning:*

✓ L_{sense}: sensing/encoding time at device.
✓ L_{tx}: network transmission time device → compute.
✓ L_{queue}: waiting time due to contention/bursts (important when "many events arise simultaneously").
✓ L_{proc}: processing time in edge/fog/cloud.
✓ L_{store}: time to persist (optional for immediate action; relevant for reliability/retention).
✓ L_{return}: control/notification back to actuator/user.

➢ *Express Transmission Time from Bandwidth + Message Size*

If payload size is S bits and available bottleneck throughput is B bits/s, with propagation/handshake overhead d:

$$L_{tx} = d + \frac{S}{B}$$

- *Why This Matches the Discussion:*

Moving Processing Closer to Sources Reduces d (Shorter Path) and Often Reduces S to the Cloud (Send Summaries Instead of Raw), Which Reduces the $\frac{S}{B}$ Part.

➢ *Compare Architectures by Plugging in Different Path Lengths*

Let d_E, d_F, d_C be "distance/handshake overhead" to edge/fog/cloud, typically:

$$d_E < d_F < d_C$$

Then (Holding All Else Equal):

$$L_{tx}(E) < L_{tx}(F) < L_{tx}(C)$$

This is exactly the "Cloud has Higher Latency; Edge/Fog are More Realistic for Low Latency Now" Framing.

➢ *Rationale and Significance of the Research*

The massive growth of connected devices featuring limited processing capabilities and the increasing frequency of data generation by these devices emphasize the importance of distributed scalable data processing capable of giving real-time insight into what is happening inside the networks. Such insight should lead the system toward a predetermined goal while detecting and preventing hazardous situations. This goal is best achieved with scalable data processing architectures that move the data processing close to the source of data generation. The consideration of both user-device proximity and hazard-full situations leads to data processing architectures that span zone 3 and zone 2 of the three-zone model, which is a common reference in the Internet of Things world. Data processing situated in these zones can provide services to both users and devices of the system that require control and automation actions based on real-time analytics.

Scalable data processing solutions that cover only these zones may lead to resource wastage. Some services, such as video surveillance, are costly in terms of bandwidth requirements and data storage resources. Furthermore, simple processing can be performed by the devices themselves, and Services that require distribution among many devices should not be processed in the edge/fog environment. In such services, the approach may be underwhelming, with response time increasing with the path delay overhead from the data source to the zone 0 cloud, even if the cloud provides more processing resources and elastic data storage. Such considerations have recently stimulated research into data processing architectures that store and analyze users' data in zone 1 and zone 0.

III. ARCHITECTURAL PARADIGMS FOR IoT DATA PROCESSING

Internet of Things (IoT) data processing is tackling rapidly growing workloads and performance requirements. Different application properties, such as scalability, latency, reliability, and data governance, lead to variant architectural paradigms. Edge processing places the analytics near the data sources to minimise latency, but at the expense of scalability and efficiency in resource utilisation. Fog computing extends this concept to provide an intermediate layer that can be leveraged by multiple data sources. Although edge and fog computing layers can exploit local resources, they usually rely on dedicated processing clusters, and therefore cannot provide the elastic scaling advantage of a cloud approach. Other architectures process the data in a central location, benefitting from the elasticity of cloud computing, but introduce latency that is conspicuous in a Large-scale Event Processing application.

IoT processing workloads and requirements often evolve in unpredicted ways, and a cloud-centric elasticity enables the effective response. Latency can also be mitigated by placing the routing nodes and data lakes close to the data sources. Different technology architecture trade-offs provide solutions to different use cases, further

enhancing the overall processing infrastructure. Such trade-off ranges are explored for a spectrum of functioning design members.

➢ *Edge Computing and Fog Computing*

Edge computing moves the entire data processing from the cloud to near the sources of data. Fog computing complements cloud services with an intermediate layer close to the edge, instead of processing traffic with the cloud only. The fog layer is less elastic, but closer to users, more responsive, and independent from the Internet. Fog-saving techniques reduce end-to-end latency and re-routing, even though all those savings may disappear for larger-scale events, when close users overpower the service capability and receive far-end service. The latency is reduced by moving the processing near the sources and by limiting the flows to the cloud. Only the raw data are sent to cloud storage to reduce traffic, to eliminate time- and space-dimension I/O and to maintain long-term reliability by taking advantage of cloud elasticity. Service traffic for fetching processed data is smaller than the edge data traffic.

Distributed sensor applications require redundant architectures to avoid faults and have higher reliability. The input flows for those fault-tolerant, and also over-scaled solutions are so great that the response time is limited by processing and not by geographical distance. Indeed, traffic is routed to the closest resource with enough capability, but this proximity does not improve end-to-end latency, because service resources do not provide it anyway. Adding fog services much increases the time- and space-dimension I/O, and with it the probability of loss, so it should be avoided when fog nodes or cloud resources are scarce. High availability of cloud resources does not prevent losses; the I/O costs must be balanced against the value of users' data. When saving traffic is more important than paying for cloud services quality-dependence should be removed, because quality depend on total processing only, not on how, where or by whom it has been done.

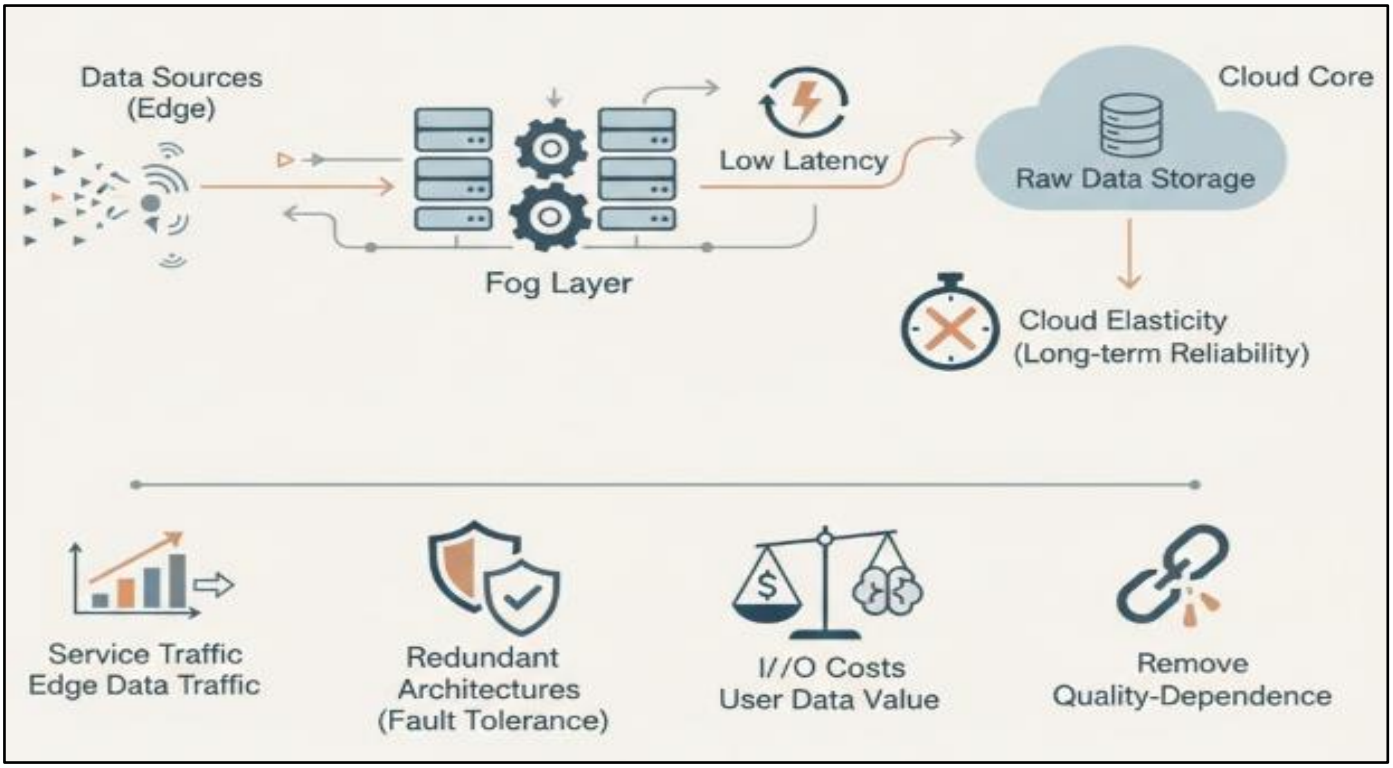

Fig 3 Optimizing the Cloud-Fog-Edge Continuum: Balancing Latency Savings, I/O Reliability, and Resource Elasticity in Distributed Sensor Networks

➢ *Cloud-Centric Architectures*

Various architectural paradigms for data processing in IoT applications may be identified, representing different trade-offs between scalability, latency, reliability, and governance. Two key paradigms are edge and fog computing, and cloud-centric architectures. The first paradigm distributes processing and storage components toward the edge of the network either in the sensors themselves, in active relays, or in nearby micro-clouds. This implementation decreases latency and the bandwidth required to communicate with the public cloud. Nevertheless, moving cloud resources to the edge enlarges the attack surface and decreases elasticity. A combined architecture exploits the benefits of both paradigms by integrating a fog layer between the cloud and the edge.

Cloud-centric architectures centralize data processing and storage resources in an elastic public cloud.

This simplifies the design of systems on top, typically relying on a secure connection to an Internet-scale Cloud Service Provider. The advantages of centralization are therefore considered alongside scalability and cost: security, availability, and long-term service provision are AWS's and Azure's business case. A cloud-centric design enables a system composed only of simple and inexpensive sensors, requiring no management other than maintenance, calibration, and possible replacement. The lack of processing and storage capabilities decreases the amount of information sent to the cloud, since the sensors can discard irrelevant information already on the edge.

Table 1 Comparison of Common IoT Communication Protocols

Architecture	Latency (lower=better)	Elastic Scalability	Resilience to Network Failure
Edge	5	2	4
Fog	4	3	4
Cloud-centric	2	5	2

IV. DATA INGESTION AND TRANSPORT MECHANISMS

Most data in modern IoT applications are transported through the air. Therefore, it is crucial to choose the right data transport mechanism for a given application scenario. This selection process may involve considerations of different aspects, including deployed protocols, data formats, and security. A wide range of existing and upcoming transport solutions are being adapted to fulfil the requirements of IoT applications while supporting interoperability through the use of adapters. Vehicle- and subscriber-publisher-based models are well served, and several stream processing frameworks aim to deal with the high-frequency nature of incoming data. Nevertheless, several challenges remain open, as new types of devices are continuously being deployed and future systems should support an ever-increasing number of connected devices. The requirements for low-latency services and the need for security, access control, and protection of sensitive user data add additional complexity. Although the adoption of message-oriented middleware provides a solution supported by several standards, there are still many situations where adequate data transport cannot be guaranteed.

Among the middleware models offered for building IoT applications, the pub/sub model plays a prominent role. Data producers or publishers send data to a shared channel, and one or more data consumers or subscribers receive messages sent to the same channel. Unlike the client-server model, where clients must be aware of the servers they are accessing, the pub/sub model hides the location of the communication partners from the involved parties, offering better dynamic scalability and flexibility when used in a distributed setting. Message-oriented middleware implementations provide built-in support for these features.

- *Protocols and Interoperability*

The choice of protocol has a direct influence on the systems architecture, and any solution requires a trade-off between throughput, structure, and capacity for maintaining interoperability. The comparison between three of the most popular IoT protocols is presented in Table 1. CoAP is a RESTful protocol designed for constrained environments and for managing the fundamental Endpoint-to-Client communication pattern, other protocols like MQTT allow for a broader range of communication schemes and are still lightweight, but are not supported in the main frameworks and platforms. Although MQTT has quickly gained prominence since its inception, it is still possible to encounter projects using EbML, especially when the ecosystem requires message partitioning. Other protocols such as DDS and AMQP may be more performant, but require greater recourses. When discussing communication at the application layer, the concept of adapter emerges, representing an application module that allows a smart object to send and receive data using its native protocol while interacting with a component, such as a cloud service, that uses a different one. The importance of adapters lies in their capacity for maintaining interoperability between systems sharing the same information model but using different protocols.

An efficient and scalable data ingestion process must support a Pub/Sub model that allows the system to present its data as a set of channels to which subscribers can express interest in receiving notifications. Because the sources of data can be smart objects, data providers, or external systems like social networks, the data volume generated can be huge and varied. Thus, the data transport layer should be designed to handle high throughput while maintaining low latency and fault tolerance. Distributed data streaming platforms such as Apache Kafka or Redis Streams are designed for such scenarios and have been adopted by the main data processing frameworks, allowing the implementation of a complete and scalable Pub/Sub model while requiring less effort in terms of maintaining the fault tolerance capacity of the communication pipeline.

- *Equation 2: Throughput and Scalability Equations (Why Cloud "Wins" Elasticity)*

The contrasts dedicated edge/fog resources with cloud elasticity.

➢ *Ingestion Rate vs Processing Capacity (Stability Condition)*

- Let incoming event rate be λ (events/s).
- Let each worker process at rate μ (events/s).
- Let there be c parallel workers (edge nodes, fog cluster nodes, or cloud instances).
- Total service capacity:

$$\mu_{\text{total}} = c\mu$$

For the system not to "fall behind":

$$\lambda < c\mu$$

- *Step-by-Step:*

✓ Each worker can do μ events/s.
✓ c workers do $c\mu$ events/s.
✓ If arrivals λ exceed that, queues explode → latency spikes (the "large-scale events overpower service capability" idea).

➢ *Elastic Scaling Effect*

Cloud elasticity means c can be increased quickly (autoscaling), so you try to keep $\lambda < c\mu$. Edge/fog have tighter ceilings on c, so they hit overload earlier even if transmission latency is low.

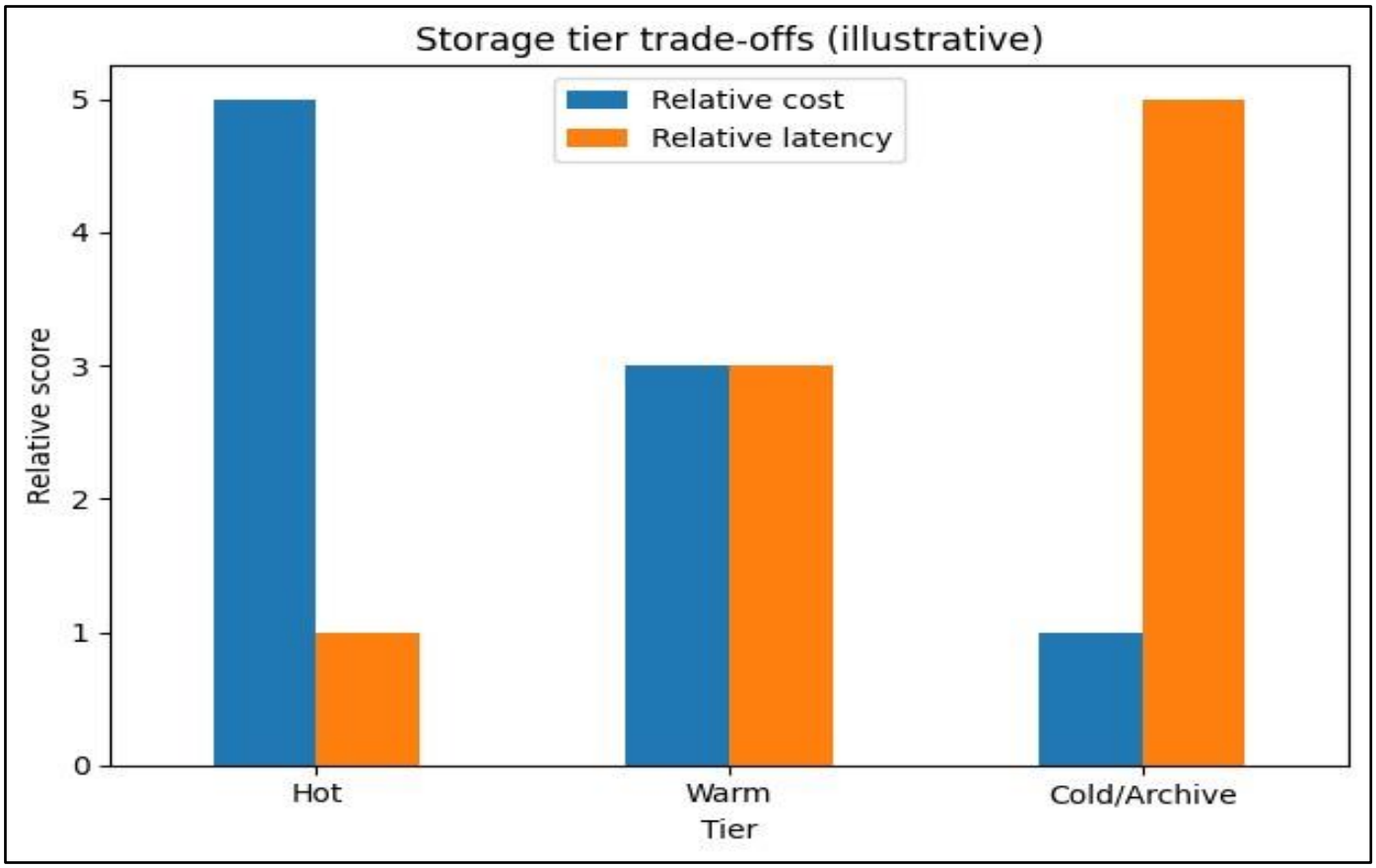

Fig 4 Stream Processing and Publish/Subscribe Framework Placement in IoT Data Pipelines

➢ *Pub/Sub and Stream Processing Frameworks*

Many IoT applications are latency-dependent, many events arise simultaneously, and device features and actions are rarely controlled by the devices' owners. Consequently, a publish/subscribe (pub/sub) model that reacts to specific events and forwards data only to interested parties minimizes both data processing and transport costs. Moreover, a data-processing framework that performs real-time analytics on these continuously generated data streams facilitates timely information delivery. The evaluation of pub/sub systems and stream-processing frameworks indicates that pub/sub systems ensure loose coupling and separation of concerns between components, and some offer polyglot persistence through the deployment of suitable adapters. Stream-processing frameworks support scalability, fault tolerance, and high throughput, but are still maturing in user-friendliness and support for strong consistency guarantees.

The need for a framework that treats all incoming data as a continuous data stream, analyzes it in real time, and produces a continuous result stream is in direct conflict with the conventional idea of batching. As the requirements of latency-sensitive data analytics become more stringent, conventional batch-processing engines that do not support streaming semantics fall short. Most stream-processing engines adopt at least a subset of the four essential characteristic features of stream-processing

systems: massive data throughput, low latency, fault-tolerant operations, and incremental processing. Such requirements exist in a wide range of applications, including telecommunication monitoring, complex-event detection, Internet-of-Things monitoring and processing, online recommendations, kitchen or fire alarms, clickstream analysis, malware detection, network intrusion detection, and realtime credit card fraud detection.

V. DATA STORAGE STRATEGIES FOR IoT WORKLOADS

Considering that the resources required to support scalable systems in terms of data storage must evolve over time according to demand, different storage strategies are required to guarantee cost-effectiveness. The most common and best described techniques for the storage of IoT workloads are time-series databases and data lakes. Time-series databases adhere to a structured data model with pre-defined schemas, optimal for the accelerated write and read access demands of sensors and actuators whose data are typically accessed by time ranges. Data lakes support several types of data throughout their life cycle, ranging from the raw stage, where the data are kept compressed for cost reduction, to the prepared stage, where subsets are optimized for high-throughput analytical queries.

While data lakes have proved to be effective storage back-ends for some IoT solutions, a time-series database is often the best option for dedicated IoT use cases managing time-series data. Any architecture dealing with a significant amount of time-series data from sensors and actuators should leverage the advantages of specialized time-series databases to ease the architecting of the overall solution. The appropriate storage-tiering strategy, intrinsic to data-lake architectures, should be accounted for in any IoT data system. Indeed, multiple copies of the same data, in distinct stages prepared for different purposes and workloads, are common for IoT implementations that can afford to store the same data more than once.

- *Time-Series Databases and Data Lakes*

Time-series databases and data lakes serve distinct roles in IoT data storage. Time-series databases optimize performance for time-dependent data, while data lakes ingest, process, and analyze data from various sources and types in their raw form. Data lake engines comment and support queries on the data and display time-series data. Time-series databases are suitable for monitoring-related tasks, while data lakes are designed for wide-range data processing and machine-learning applications. Data ingestion speed is key for monitoring, which is why time-series databases outperform data lakes when the query frequency is high. The ingestion speed of data lakes is lower, but they also provide storage for data in its raw format, which can be queried and processed later.

Time-series data stored in a data lake can be reduced in cost and volume by using time-series data management techniques. They include data lifecycle management, where temporally specific policies enable a compression procedure to control the data volume; tiered storage, where older data moves from faster access storage to low-cost storage; and direct data fetching from the raw data. Therefore, a data lake acts as the primary storage for IoT data and an additional storage layer for dynamically compressed, older time-series data.

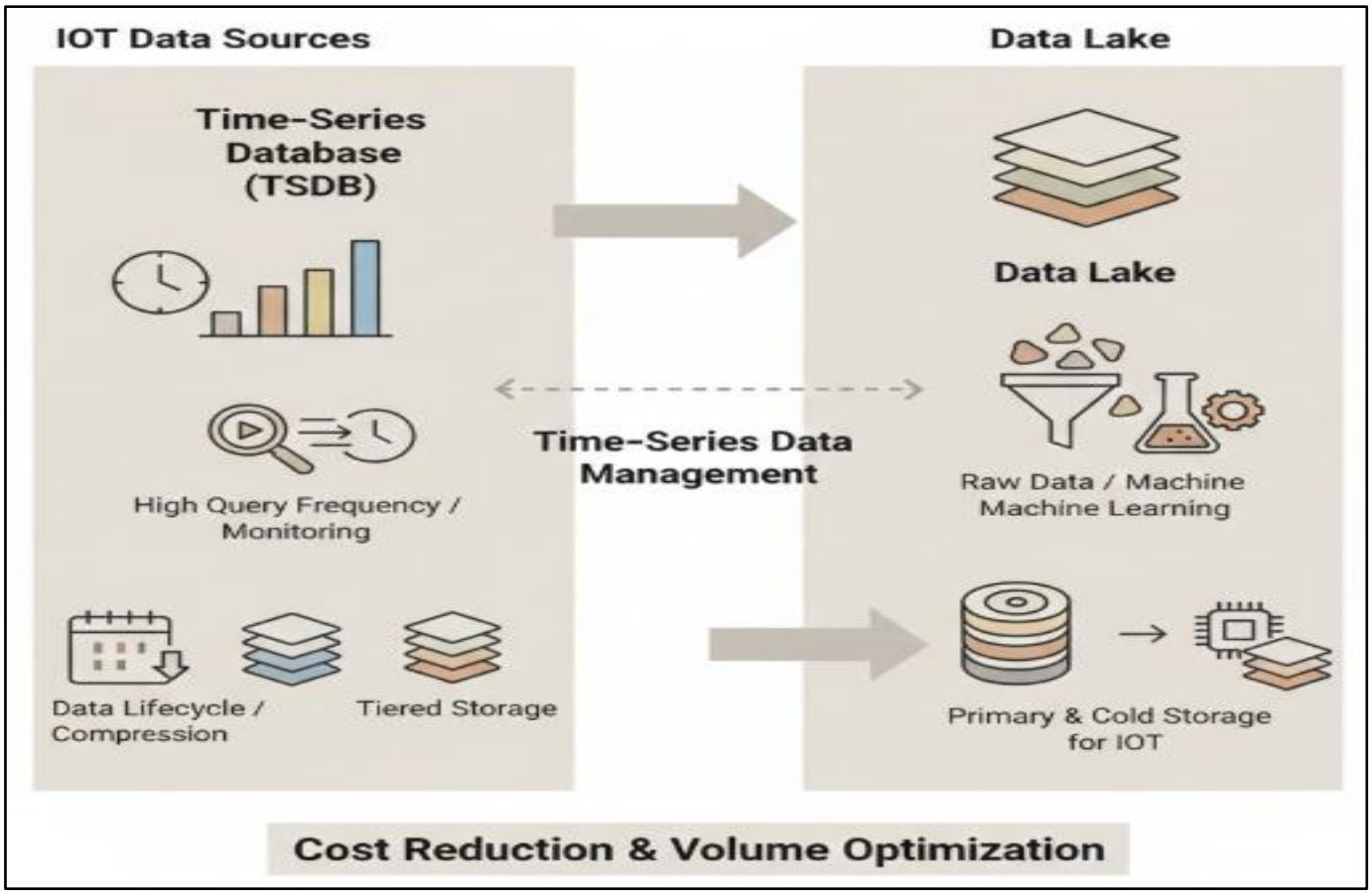

Fig 5 A Unified Tiered Storage Architecture for IoT: Integrating Time-Series Databases and Data Lakes for Optimized Monitoring and Lifecycle Management

➢ *Storage Tiering and Retention Policies*

The lifecycle of IoT data typically involves three distinct territories: a hot phase for a small percentage of recent data used for operational decision-making, a warm phase for larger volumes of data stored at lower costs, and a cold or archival phase. Different types of data may have different cost and performance requirements, and a suitable storage tiering strategy reduces costs while retaining a balance between the two.

Recent research has shown that time-series data in different timescales can be modeled by a joint distribution with skewed marginal distribution. Because the bulk of the data in sampled time-series exhibits an asymptotic property, the data can be adequately compressed applying a joint data-adaptive process. To speed up the compression, the data in hot and warm tier of the time series should not be compressed. This motivates the deployment of a retention policy for time-series data in all three tiers.

Edge and fog solutions have a limited lifetime for stored data due to the finite storage space and frequent changes in context (for instance, the relocation of a device). However, publishing a notification announcing the imminent deletion of data allows decision-makers to trigger the remote transfer of high-value data to a central data store. In an initial phase, the warm tier of the time-series database may keep a full resolution (but uncompressed) version of the series over a shorter time span, to be down sampled later on. The approach can also guide the creation of archival policies for historical analytic storage systems, specifying different compression algorithms and retention periods, due to the asymptotic properties explicitly tested in the evaluations of time-series model and joint-adaptive compression.

Table 2 Performance Comparison of Edge, Fog, and Cloud-Centric Architectures Across Key Metrics

Protocol	Style	Typical Fit	Overhead
CoAP	RESTful (request/response)	Constrained devices, endpoint-client	Low
MQTT	Pub/Sub (brokered)	Lightweight telemetry/events	Low
DDS	Pub/Sub (data-centric)	High-performance, real-time systems	Higher
AMQP	Messaging (brokered)	Enterprise messaging / reliability	Higher

VI. STREAM PROCESSING AND REAL-TIME ANALYTICS

Intelligent data processing not only has processing times measured in milliseconds, but it also echoes the temporal semantics outlined in the application requirements. The support for temporal semantics naturally leads to real-time and stream processing. Stream processing keeps data in a transient state through an event-based architecture; intelligent data processing is built on event-driven architectures. The central performance characteristics of a stream processing framework are Loop throughput, latency, and fault tolerance. Latency can be introduced in two ways: event processing and system latency. Modern streams processing frameworks make use of the concept of Window to divide the continuous data flow into smaller and more manageable parts.

Data arriving in streams can be processed as soon as they arrive to provide near real-time responses to end users or other systems. Given the non-persistent nature of stream data, the processing serves a transient purpose for immediate actions. Nevertheless, this doesn't mean that every arriving record will be acted upon, processed requests need to be made according to business requirements. In a practical deployment, not every data record could possibly be stored in every set of systems. The concept of Window helps to operate upon a smaller subset of data rather than process the entire set of data stored in the stream. The purpose of Window is to break the continuously growing stream of data into a finite batch of data ensuring that only the required data is processed to achieve the business requirement.

Windowing can be done over count or time. Count Window triggers an action every k number of records whereas Time Window executes an action every millisecond. Stateful processing allows event correlation, the implementation of business rules for common patterns in the stream, grouping of related events to calculations and large event patterns such as fraud detection. Stateful processing enables frameworks to remember stateful information for better analysis. The nature of Window also implies that the state maintained by stream processing framework is ultra-short. When processing streaming data it might be acceptable to lose processing capability on a couple of events – something that can be achieved by using an At-least-once processing guarantee. However, in traditional systems from the enterprise world, losing events can create huge issues. Many companies choose a message broker with exactly-once guarantee.

- *Equation 3: Pub/Sub Fan-Out Cost Equations (Ingestion/Transport)*

The emphasizes pub/sub for decoupling and scalability, and Kafka/Redis Streams as platforms.

➢ *Producer → Broker → Subscribers' Traffic*

Let:

- Publishers produce at rate λ events/s,
- Average payload S bits,
- Number of subscribers on a topic is m.

- *Without Broker-Side Filtering/Aggregation*, network egress from broker is approximately:

$$T_{\text{egress}} \approx \lambda \cdot m \cdot S$$

- *Step-by-Step:*

✓ Each event is S bits,
✓ λ events each second,
✓ Sent to m subscribers → multiply by m.

This motivates selective subscriptions + event filtering ("forward only to interested parties minimizes processing and transport costs").

➢ *Benefit of Filtering/Aggregation at Edge/Fog*

If edge/fog reduces payload by factor r (e.g., summaries), where $0 < r < 1$, then to cloud:

$$S_{\text{to cloud}} = rS \quad \Rightarrow \quad T_{\text{uplink}} = \lambda \cdot rS$$

So, traffic reduction ratio is:

$$\frac{T_{\text{uplink}}}{\lambda S} = r$$

➢ *Windowing, State Management, and Exactly-Once Semantics*

In stream processing, windowing is a fundamental operation for handling infinite streams by breaking them into finite segments. It allows applying an aggregation operation to a portion of the stream, enabling operations like counting events over the past minute or calculating the average temperature in the last hour. The choice of windowing strategy is influenced by the operations applied to the data.

A stateful operation is one that preserves state across invocations. Maintaining state leads to fault tolerance challenges, as reprocessing events from the beginning of the stream could require excessive time and resources. A distributed stream processing framework builds a directed acyclic graph (DAG) for the job, utilizing external systems like Apache ZooKeeper for consensus and failover. For stateful operators, partitions for each stream are replicated, noting the last processed event in a change log. If an instance fails, a backup instance can start reading from the last checkpoint and use the replayed events from the change log to reconstruct its state.

Exactly-once semantics guarantees that each input event triggers exactly one invocation of the stream operator, allowing the system to appear as if it were executed in isolation, even with failures and retries. However, using two-phase commits leads to data transfer overhead, affecting throughput. Achieving exactly-once semantics is a key question in stream processing, and either the input source or the output sinks must be prepared for it.

VII. CONCLUSION

Architectural trade-offs for data-intensive Internet of Things applications are evaluated and categorized according to their suitability for minimizing latency, maximizing throughput, ensuring reliability, and enabling proper governance, thus addressing the problems faced by data-centric systems. The analysis considers data ingestion and transport modes, storage handling of IoT workloads, and real-time processing capabilities. It integrates the requirements of different architecture layers—namely edge, fog, cloud, and their combination—documents why no solution fulfills all requirements, and surveys service- and data-centric alternatives for high-throughput, low-latency, fault-tolerant, reliable processing of large amounts of data.

Current architectures cannot cover all data-related objectives: low-latency edge and fog computing suffer from limited resources and strict but irregular data-flow control, whereas cloud-centric approaches are best placed for subsecond latencies but operate at high, varying throughputs. Scaling through a wider geographical distribution of clouds maintains these advantages but complicates costs and governance. For pushing latency beyond a few seconds and coping with data steam-heavy workloads, the service-centric model is best applied; presented here without reference to the underlying processing framework, it relies on publish-and-subscribe communication to isolate system elements. Although not yet widely established, a data-centric perspective increases throughput and fault tolerance above that of traditional stream-processing modules.

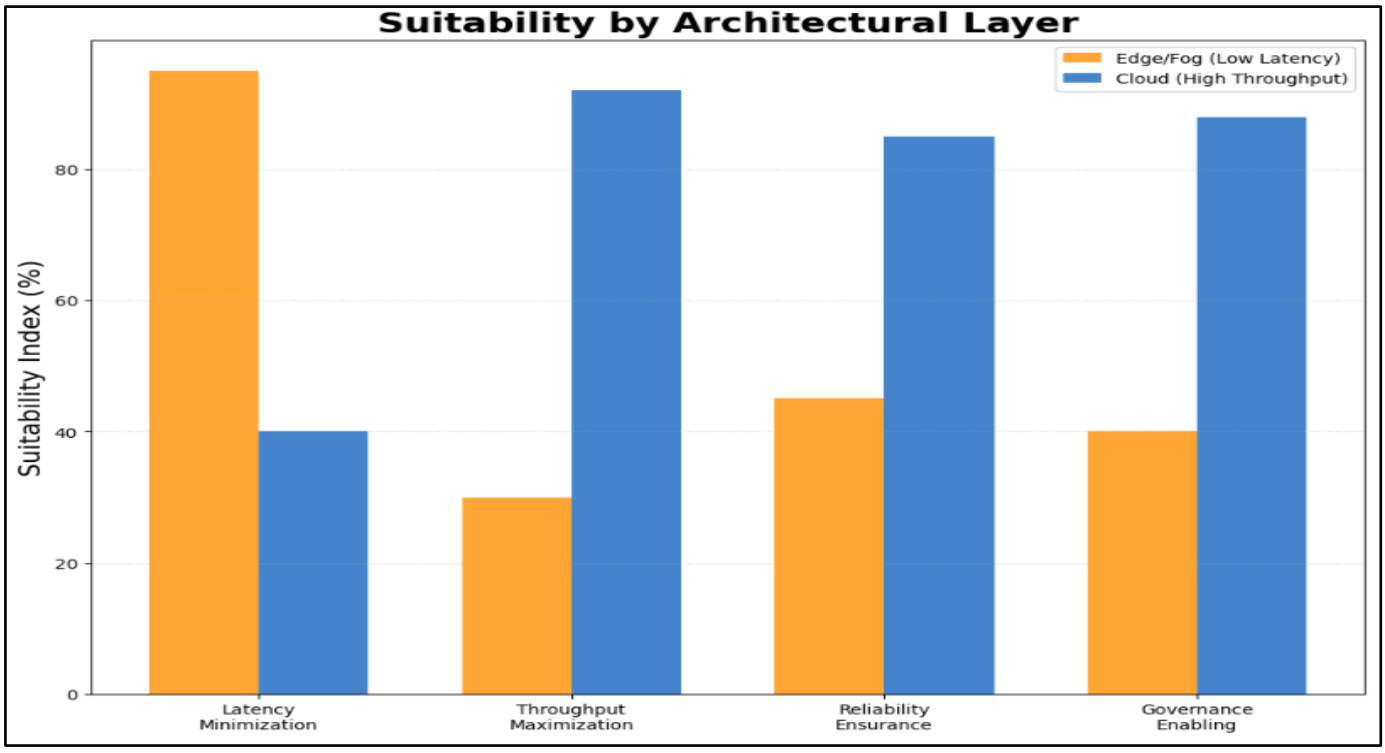

Fig 6 Suitability by Architectural Layer

➢ *Summary and Future Directions*

Scalable Data Processing Architectures for Internet of Things Applications (2022).

Data processing for Internet of Things applications must scale effectively and incorporate ever-increasing device use cases, services, and business logic into data flows. The space of Scandinavian and Finnish data processing for IoT applications is rich in diverse forms of edge, fog, and cloud-centric models. Various architectural characteristics must be considered: management and governance, throughput and latency, topology and fault tolerance, and scaling. Practical architectures are subsequently identified for a representative set of IoT use cases.

As the scale of data ingress and egress grows, data transport no longer occupies a purely background role. A multitude of different protocols is used and proposed at various layers of the OSI model, with different

characteristics and styles of operation. Providing support for data interchange between diverse protocols providing pub/sub or event-oriented features is important in virtually every aspect of the realisation of IoT. The operational consideration of the long-running processes affecting storage and archiving have made data lakes popular. Nevertheless, the fact that in IoT applications time-series data dominates has put time-series databases in a leading position, providing purpose-built services and support features for managing them.

Streaming analytics and the real-time aspect of IoT systems have continued to gain attention. Approaches used in the area can be found in research devoted at the level of windowing, fault tolerance, exactly-once semantics, redundant event sources and sinks, condition verification, and information evolution. Regardless of the exact service and bus being considered, distribution and properly sizing for the workload have to be treated as a first-class concern.

REFERENCES

[1]. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, 2013.

[2]. Dwaraka Nath Kummari. (2022). Fiscal Policy Simulation Using AI And Big Data: Improving Government Financial Planning. Kurdish Studies, 10(2), 934–945. https://doi.org/10.53555/ks.v10i2.3855

[3]. Atzori, L., Iera, A., Morabito, G. The Internet of Things: A survey. Computer Networks, 2010. Journal of Interna-tional Crisis and Risk Communication Research, 168-193.

[4]. Goutham Kumar Sheelam, "Semiconductor Innovation for Edge AI: Enabling Ultra-Low Latency in Next-Gen Wireless Networks," International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE), DOI: 10.17148/IJARCCE.2022.111258.

[5]. Borgia, E. The Internet of Things vision: Key features, applications and open issues. Computer Communications, 2014.

[6]. Meda, R. (2022). Integrating Edge AI in Smart Factories: A Case Study from the Paint Manufacturing Industry. International Journal of Science and Research (IJSR), 1473-1489.

[7]. Zanella, A., Bui, N., Castellani, A., Vangelista, L., Zorzi, M. Internet of Things for smart cities. IEEE Internet of Things Journal, 2014.

[8]. Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D. Context aware computing for the Internet of Things: A survey. IEEE Communications Surveys & Tutorials, 2014.

[9]. Inala, R. (2022). Engineering Data Products for Investment Analytics: The Role of Product Master Data and Scalable Big Data Solutions. International Journal of Scientific Research and Modern Technology, 155-171.

[10]. Stankovic, J. A. Research directions for the Internet of Things. IEEE Internet of Things Journal, 2014.

[11]. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M. Internet of Things: A survey on enabling technologies, protocols, and applications. IEEE Communications Surveys & Tutorials, 2015.

[12]. Garapati, R. S. (2022). Web-Centric Cloud Framework for Real-Time Monitoring and Risk Prediction in Clinical Trials Using Machine Learning. Current Research in Public Health, 2, 1346.

[13]. Nagabhyru, K. C. (2022). Bridging Traditional ETL Pipelines with AI Enhanced Data Workflows: Foundations of Intelligent Automation in Data Engineering. Available at SSRN 5505199.

[14]. Mineraud, J., Mazhelis, O., Su, X., Tarkoma, S. A gap analysis of Internet-of-Things platforms. Computer Communications, 2016.

[15]. Avinash Reddy Aitha. (2022). Deep Neural Networks for Property Risk Prediction Leveraging Aerial and Satellite Imaging. International Journal of Communication Networks and Information Security (IJCNIS), 14(3), 1308–1318. Retrieved from https://www.ijcnis.org/index.php/ijcnis/article/view/8609.

[16]. Ray, P. P. A survey on Internet of Things architectures. Journal of King Saud University – Computer and Information Sciences, 2018.

[17]. Gottimukkala, V. R. R. (2022). Licensing Innovation in the Financial Messaging Ecosystem: Business Models and Global Compliance Impact. International Journal of Scientific Research and Modern Technology, 1(12), 177-186.

[18]. Botta, A., de Donato, W., Persico, V., Pescapé, A. Integration of cloud computing and Internet of Things: A survey. Future Generation Computer Systems, 2016.

[19]. Satyanarayanan, M. The emergence of edge computing. Computer, 2017.

[20]. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L. Edge computing: Vision and challenges. IEEE Internet of Things Journal, 2016.

[21]. Segireddy, A. R. (2020). Cloud Migration Strategies for High-Volume Financial Messaging Systems.

[22]. Bonomi, F., Milito, R., Natarajan, P., Zhu, J. Fog computing: A platform for Internet of Things and analytics. In: Big Data and Internet of Things: A Roadmap for Smart Environments, 2014.

[23]. Amistapuram, K. (2022). Fraud Detection and Risk Modeling in Insurance: Early Adoption of Machine Learning in Claims Processing. Available at SSRN 5741982.

[24]. Yi, S., Qin, Z., Li, Q. Security and privacy issues of fog computing: A survey. In: Wireless Algorithms, Systems, and Applications (WASA), 2015.

[25]. Chiang, M., Zhang, T. Fog and IoT: An overview of research opportunities. IEEE Internet of Things Journal, 2016.

[26]. Rongali, S. K. (2022). AI-Driven Automation in Healthcare Claims and EHR Processing Using MuleSoft and Machine Learning Pipelines. Available at SSRN 5763022.

[27]. Gupta, H., Vahid Dastjerdi, A., Ghosh, S. K., Buyya, R. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments. Software: Practice and Experience, 2017.

[28]. Varghese, B., Wang, N., Barbhuiya, S., Kilpatrick, P., Nikolopoulos, D. J. Challenges and opportunities in edge computing. In: IEEE International Conference on Smart Cloud (SmartCloud), 2016.

[29]. Roman, R., Lopez, J., Mambo, M. Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. Future Generation Computer Systems, 2018.

[30]. Varri, D. B. S. (2022). AI-Driven Risk Assessment and Compliance Automation in Multi-Cloud Environments. Available at SSRN 5774924.

[31]. Mouradian, C., et al. A comprehensive survey on fog computing: State-of-the-art and research challenges. IEEE Communications Surveys & Tutorials, 2018.

[32]. Mahmud, R., Koch, F. L., Buyya, R. Cloud-fog interoperability in IoT-enabled healthcare solutions. In: International Conference on Distributed Computing Systems (ICDCS) Workshops, 2016.

[33]. Vadisetty, R., Polamarasetti, A., Guntupalli, R., Raghunath, V., Jyothi, V. K., & Kudithipudi, K. (2022). AI-Driven Cybersecurity: Enhancing Cloud Security with Machine Learning and AI Agents. Sateesh kumar and Raghunath, Vedaprada and Jyothi, Vinaya Kumar and Kudithipudi, Karthik, AI-Driven Cybersecurity: Enhancing Cloud Security with Machine Learning and AI Agents (February 07, 2022).

[34]. Dean, J., Ghemawat, S. MapReduce: Simplified data processing on large clusters. Communications of the ACM, 2008.

[35]. Zaharia, M., et al. Spark: Cluster computing with working sets. In: USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2012.

[36]. Zaharia, M., et al. Discretized streams: Fault-tolerant streaming computation at scale. In: ACM Symposium on Operating Systems Principles (SOSP), 2013.

[37]. Gottimukkala, V. R. R. (2021). Digital Signal Processing Challenges in Financial Messaging Systems: Case Studies in High-Volume SWIFT Flows.

[38]. Carbone, P., et al. Apache Flink: Stream and batch processing in a single engine. IEEE Data Engineering Bulletin, 2015.

[39]. Toshniwal, A., et al. Storm@Twitter. In: ACM SIGMOD International Conference on Management of Data, 2014.

[40]. Dwaraka Nath Kummari,. (2022). Machine Learning Approaches to Real-Time Quality Control in Automotive Assembly Lines. Mathematical Statistician and Engineering Applications, 71(4), 16801–16820. Retrieved from https://philstat.org/index.php/MSEA/article/view/2972

[41]. Kreps, J., Narkhede, N., Rao, J. Kafka: A distributed messaging system for log processing. In: NetDB Workshop, 2011.

[42]. Akidau, T., et al. The Dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. Proceedings of the VLDB Endowment, 2015.

[43]. Goutham Kumar Sheelam. (2022). Reconfigurable Semiconductor Architectures For AI-Enhanced Wireless Communication Networks. Kurdish Studies, 10(2), 1027–1040. https://doi.org/10.53555/ks.v10i2.3867

[44]. Vavilapalli, V. K., et al. Apache Hadoop YARN: Yet another resource negotiator. In: ACM Symposium on Cloud Computing (SoCC), 2013.

[45]. Shvachko, K., Kuang, H., Radia, S., Chansler, R. The Hadoop Distributed File System. In: IEEE MSST, 2010.

[46]. Meda, R. (2020). Designing Self-Learning Agentic Systems for Dynamic Retail Supply Networks. Online Journal of Materials Science, 1(1), 1-20.

[47]. [Lakshman, A., Malik, P. Cassandra: A decentralized structured storage system. ACM SIGOPS Operating Systems Review, 2010.

[48]. Inala, R. (2021). A New Paradigm in Retirement Solution Platforms: Leveraging Data Governance to Build AI-Ready Data Products. Journal of International Crisis and Risk Communication Research, 286-310.
[49]. Fowler, M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.
[50]. Marz, N., Warren, J. Big Data: Principles and best practices of scalable real-time data systems. Manning, 2015.
[51]. Aitha, A. R. (2022). Cloud Native ETL Pipelines for Real Time Claims Processing in Large Scale Insurers. Available at SSRN 5532601.
[52]. Kleppmann, M. Designing Data-Intensive Applications. O'Reilly Media, 2017.
[53]. Hohpe, G., Woolf, B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, 2003.
[54]. Gottimukkala, V. R. R. (2020). Energy-Efficient Design Patterns for Large-Scale Banking Applications Deployed on AWS Cloud. power, 9(12).
[55]. Stonebraker, M., et al. The end of an architectural era: (It's time for a complete rewrite). In: VLDB, 2007.
[56]. Avinash Reddy Segireddy. (2022). Terraform and Ansible in Building Resilient Cloud-Native Payment Architectures. International Journal of Intelligent Systems and Applications in Engineering, 10(3s), 444–455. Retrieved from https://www.ijisae.org/index.php/IJISAE/article/view/7905.
[57]. Abadi, D. J. Data management in the cloud: Limitations and opportunities. IEEE Data Engineering Bulletin, 2009.
[58]. Chang, F., et al. Bigtable: A distributed storage system for structured data. In: USENIX OSDI, 2006.
[59]. Keerthi Amistapuram , "Energy-Efficient System Design for High-Volume Insurance Applications in Cloud-Native Environments," International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering (IJIREEICE), DOI 10.17148/IJIREEICE.2020.81209
[60]. Armbrust, M., et al. A view of cloud computing. Communications of the ACM, 2010.
[61]. Kliazovich, D., Bouvry, P., Khan, S. U. GreenCloud: A packet-level simulator of energy-aware cloud computing data centers. Journal of Supercomputing, 2012.
[62]. Cavanagh, J. F. The Kappa Architecture. (Industry architecture concept widely attributed pre-2016; commonly documented in conference talks and technical writings prior to 2022).
[63]. Rongali, S. K. (2020). Predictive Modeling and Machine Learning Frameworks for Early Disease Detection in Healthcare Data Systems. Current Research in Public Health, 1(1), 1-15.
[64]. Zhang, Y., Chen, M., Mao, S., Hu, L., Leung, V. C. M. CAP: Community activity prediction based on big data analysis for smart cities. IEEE Network, 2014.
[65]. Chen, M., Mao, S., Liu, Y. Big data: A survey. Mobile Networks and Applications, 2014.
[66]. Khan, W. Z., et al. Big data challenges and opportunities in the hype of Industry 4.0. IEEE Communications Magazine, 2017.
[67]. Varri, D. B. S. (2022). A Framework for Cloud-Integrated Database Hardening in Hybrid AWS-Azure Environments: Security Posture Automation Through Wiz-Driven Insights. International Journal of Scientific Research and Modern Technology, 1(12), 216-226.
[68]. Jain, P., et al. A survey of big data analytics for healthcare. Journal of Big Data, 2019.
[69]. Hwang, K., Dongarra, J., Fox, G. C. Distributed and Cloud Computing: From Parallel Processing to the Internet of Things. Morgan Kaufmann, 2012.
[70]. Vadisetty, R., Polamarasetti, A., Guntupalli, R., Rongali, S. K., Raghunath, V., Jyothi, V. K., & Kudithipudi, K. (2021). Legal and Ethical Considerations for Hosting GenAI on the Cloud. International Journal of AI, BigData, Computational and Management Studies, 2(2), 28-34.
[71]. Yandamuri, U. S. (2022). Big Data Pipelines for Cross-Domain Decision Support: A Cloud-Centric Approach. International Journal of Scientific Research and Modern Technology, 227. https://doi.org/10.38124/ijsrmt.v1i12.1111
[72]. ISO/IEC 20922:2016. Information technology — Message Queuing Telemetry Transport (MQTT) v3.1.1. ISO/IEC Standard, 2016.
[73]. Shelby, Z., Hartke, K., Bormann, C. The Constrained Application Protocol (CoAP). IETF RFC 7252, 2014.
[74]. Banks, A., Gupta, R. MQTT Version 3.1.1. OASIS Standard, 2014.