A Data Brokering Architecture to Guarantee Nonfunctional Requirements in IoT Applications

Taha Mansouri[1](✉), Julian M. Bass[1], Tarek Gaber[1], Steve Wright[2], and Benedict Scorey[3]

[1] University of Salford, Manchester, UK
t.mansouri@salford.ac.uk
[2] Invisible Systems Ltd, Manchester, UK
[3] Forsberg Services Ltd, Morecambe, UK

Abstract. IoT sensors capture different aspects of the environmental data and generate high throughput data streams. To harvest potential values from these sensors, a system fulfilling the big data requirements should be designed. In this work, we reviewed the important nonfunctional requirements, in particular big data-based ones. Moreover, we dug out a conventional IoT architecture to address these requirements. Finally, we designed a brokering based architecture which is flexible and scalable enough to cover big data requirements of high throughput data streams resulted from modern sensors. Evaluation results using quantitative comparisons on use case displayed that the proposed new architecture outperformed the conventional ones. The experiments showed that the proposed architecture can handle 32 times more load than the conventional.

Keywords: IoT · wireless sensors · software architecture · data brokering

1 Introduction

The Internet of Things is a network of various devices that exchange data and communicate with each other over the Internet based on a contract agreement (Yue and He 2018). This dynamic network is available almost anywhere (Poniszewska-Maranda and Kaczmarek 2015). IoT unify the physical and the virtual domains using the Internet (Balaji et al. 2019). Systems built using the concept of IoT are based not only on the simple sensors that transmit information to the systems but also operate primarily based on statistics and simple mathematical calculations. In IoT, data integrations over different environments are thus challenging and will be supported by modular, interoperable components. Therefore, architectures should be open and follow standards; they should not restrict users from using fixed, end-to-end solutions (Shah and Yaqoob 2016; Shanzhi Chen et al. 2014). Meanwhile, IoT architectures should consider big data requirements, including volume, velocity, value, variety, and veracity (Farhan et al. 2017).

Besides functional requirements coupled with any specific application, important non-functional requirements are common in most IoT applications. Security and privacy

R. Hou et al. (Eds.): BDTA 2021/2022, LNICST 480, pp. 75–84, 2023.
https://doi.org/10.1007/978-3-031-33614-0_6

encompass security mechanisms (Aksu et al. 2018) to preserve users' crucial data and ensure their integrity alongside the architecture. It includes maintaining confidentiality, access control, and policy enforcement such as encryption algorithms (Tunc et al. 2021). Scalability denotes how the architecture can handle the constant streams of new devices or changes in throughput of the existing ones (Aksu et al. 2018). Interoperability happens among different entities. It is critical for communicating information across devices, layers, functions, and applications (Shanzhi Chen et al. 2014). Fault tolerance denotes how resilient the architecture is in case of faults in constituent entities (Farhan et al. 2017). In other words, whether, if a failure occurred in one component of the architecture, the whole system would continue functioning or not. Availability identifies the number of computing resources available to implement IoT applications (Aksu et al. 2018). And finally fast response is how much latency end users experience before receiving a response to their request (Afzal et al. 2019).

Table 1 summarizes the most important non-functionals reported in the literature. The table implies that security and scalability are the most important ones. Moreover, IoT has all notions of big data, including volume of data, variety of data types, the velocity of data streams, and value of the stored data. In terms of the volume, the problem is with storing and managing huge amounts of received data for long periods. Regarding the need to handle variety, there is an increasing requirement to communicate through a growing number of platforms, a growing number of protocols, new sensors with cutting edge technologies, and some standards. This influences the data types and data structures being processed and stored. With the growth of the number of sensors, customers, applications, and more rapid networks, the throughputs of incoming data are increasing, making retrieving the same data from conventional databases and extracting added value increasingly challenging. These sensors sense different aspects of the environment; therefore, processing them to extract valuable knowledge and insight is more demanding. Therefore, real-time analytics is a serious requirement.

Table 1. Important non-functional requirements for designing an IoT architecture.

Requirement	References
Security and Privacy	(Afzal et al. 2019; Aksu et al. 2018; Balaji et al. 2019; Farhan et al. 2017; Shah and Yaqoob 2016; Shanzhi Chen et al. 2014; Tunc et al. 2021)
Scalability	(Afzal et al. 2019; Balaji et al. 2019; Farhan et al. 2017; Shah and Yaqoob, 2016; Shanzhi Chen et al. 2014; Tunc et al. 2021)
Interoperability	(Afzal et al. 2019; Balaji et al. 2019; Farhan et al. 2017; Shanzhi Chen et al. 2014; Tunc et al. 2021)
Fault tolerance	(Balaji et al. 2019; Farhan et al. 2017; Tunc et al. 2021)
Availability	(Shah and Yaqoob 2016; Tunc et al. 2021)
Fast response/Low latency	(Afzal et al. 2019; Balaji et al. 2019; Shah and Yaqoob 2016; Tunc et al. 2021)

There are other important requirements such as the expectations of customers to push data into other third-party platforms, varying from large players like Microsoft Azure and Amazon AWS to smaller bespoke systems over various APIs. This work aims to design a scalable IoT architecture to ensure the important non-functional requirements in particular big data requirements. Therefore, the research question is which architecture is more suitable to guarantee big data requirements. The rest of this paper will consider the existing multi-tier IoT architecture and the potential improvements. Then we describe the new architecture and explain how it will tackle the non-functional requirements.

2 Existing Architecture

IoT service providers usually adopt a generic architecture and evolve it over years. As time has gone on, the number of different sensors, their applications, and the breadth of wireless communication protocols used for this. In order to visualize the data, perform analysis, and provide event-based actions and notifications for their customers, these companies either develop their own platforms or use the existing cloud-hosted services. The conventional architecture includes a very large RDBMS backend database and a web front end. The components of this architecture are tightly coupled together to conduct the main functionalities. Which is to ingest data to the backend database and disseminate them through the web applications and third parties. Figure 1 highlights the main components of this architecture.

In this architecture device is primarily a wireless sensor to capture different aspects of the environment, such as humidity, temperature, state, vibration, and so on, and transmit them through a network. The devices communicate through different protocols, including LoRa, LoRaWAN, and NB-IoT. Devices with LoRa and LoRaWAN types use a gateway which is a connected appliance to connect to the other components. However, NB-IoT sensors send data directly to the uplink network. Therefore, the communication protocols and the data types are different. Due to the different devices, there is a heavy weighted and tightly coupled ingestion layer in this architecture, which is in charge of carrying out most receiving data packets, returning acknowledgment, decryption and decoding packet data, and storing them in a local RDBMS. Then other processes running in this layer extract data from the local database, transform them to give them applicable schemas and load them into the main database. Moreover, this layer processes data sharing requests raised by third parties, fetching data from the main database and sending them an agreed protocol.

The primary storage is an RDBMS containing a few transactional tables that grow fast regarding incoming data streams' throughput. Moreover, many dimensional tables depend on the changes of the incoming data structures and the web application requirements. Finally, the web front end and a core of APIs interact with this database to fetch and share data and save the related commands and configurations.

While the architecture denoted in Fig. 1 could cover the many functional requirements, changes in the business environment and the fast-growing non-functional requirements, there are some improvements to be considered to make the architecture more manageable. Based on the big data 4 Vs (Farhan et al. 2017), we have classified these improvements into the followings:

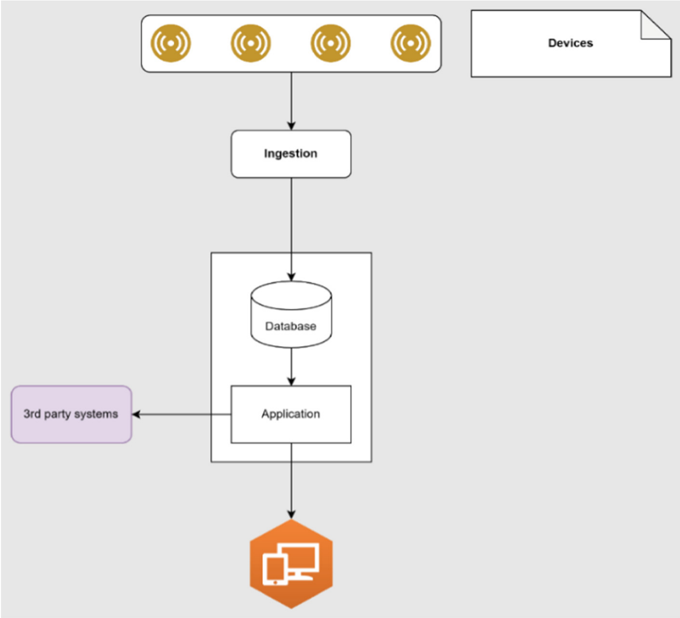

Fig. 1. A conventional IoT architecture containing a layer to gather data from different sources with different protocols, a database which is mostly a relational database, and an application as the serving layer.

2.1 Volume

Many companies, as time passes the number of customers and the period that data has been accumulated have both grown. Database changes over the years and upgrades to the database engine have all helped. It leads to a few very large tables for which guaranteeing the performance is becoming more challenging. Fast fetching data and parallel access are common problems in this environment. Moreover, managing this volume of concentrated data, such as running backups, is getting harder. Therefore, a solution needs to be more scalable, fault tolerant, with fast response and low latency data access.

2.2 Variety

In the common multi-tiered architectures, the backend data storage is still tied very heavily to the product range as it was in the early years of development. This has its consequences on the scalability and interoperability of the architecture. Early development focuses on a relatively narrow set of products. The main tables, stored in an RDBMS database, have a flat structure with a predefined number of fields for all sensors. For sensors that report less than this, it results in empty fields, whereas for sensors that report more, or where the values will not fit in the number and type available, an awkward arrangement involving several rows is required. Whilst traditional database management systems using Structured Query Languages guarantee non-functional requirements such as simplicity, consistency, availability, and performance, they suffer from static schemas and consequently limited applications for IoT (Rautmare and Bhalerao 2016).

With the development of sensors with several measurands, and the ingress of data from third-party sensors and platforms, the frequency of awkward workarounds grows, and the system becomes harder to maintain. Moreover, since all components, such as the heavily weighted ingestion layer, the database, and the web application, are tightly coupled, and any change is a radical refactor.

2.3 Velocity

With the evolution of technology, and the rise of powerful competing platforms – many developed by big names such as Microsoft in their Azure IoT platform – expectations have moved on. Expectations are now to see current 'live' data streamed to cloud platforms and dashboards, where live now means within a few seconds. Although technologies such as web-sockets could be used to deliver data to the user interface, it is difficult to see how the back-end structure could support this, given the current architecture and speed issues. To keep pace with evolving technologies, a new approach is needed to improve horizontal scalability, availability, fast response, and low latency.

Key to achieving high velocity delivery of data through the system is event driven data processing and handling. Modern transport layer technologies make continuous data streaming from/to many clients a viable option with little overhead. The common platform uses usually batching techniques which are not event driven, increasing latency and making the system unreactive to data changes.

2.4 Value Extraction

With the growing size of the data storage tables in the database comes the associated slowing down of queries that pull from this data. A general approach is to query and process stored data at the point of use. However, time taken to query data grows to such a length that it is impossible to display live real-time values of this aggregated data in some cases. And even where it is still possible, on sites with many such sensors, or where this is an overview of many customer sites, it becomes impossible. The above speed issues always restrict any further analytics of data. Therefore, the current architecture with one very large store of raw sensor data is starting to limit future possibilities and undermine system performance to the point where it becomes a serious business risk.

3 The Proposed New Architecture

In order to mitigate the mentioned challenges and to guarantee the important nonfunctional requirements, a new architecture is required. The main goal is to move the existing web platform from being at the heart of the system to becoming one of the potentially many subscribers to data in a brokered architecture and enabling big data requirements. The idea is to leverage a broker-based architecture. Figure 2 highlights the most important components of the new architecture which is based on a microservice architectural model. All existing data ingress from devices is seen as data feeders to the broker in this architecture. Future ingress from other devices, APIs, or elsewhere is further data feeders adhering to a well-defined protocol. Eexisting web applications are subscribers and sit alongside other cloud platforms or connectors subscribed to the broker; the exchange of information with the broker follows the same defined rules and protocols available to other platforms. Configuration and other such activities could be done by the web application but would not have to be done by it; other platforms could do the same via the broker and a well-defined protocol. Where data is only going to a third party, it would never touch the web application cloud platform; the data feeders and brokers only forward data to authenticated subscribers.

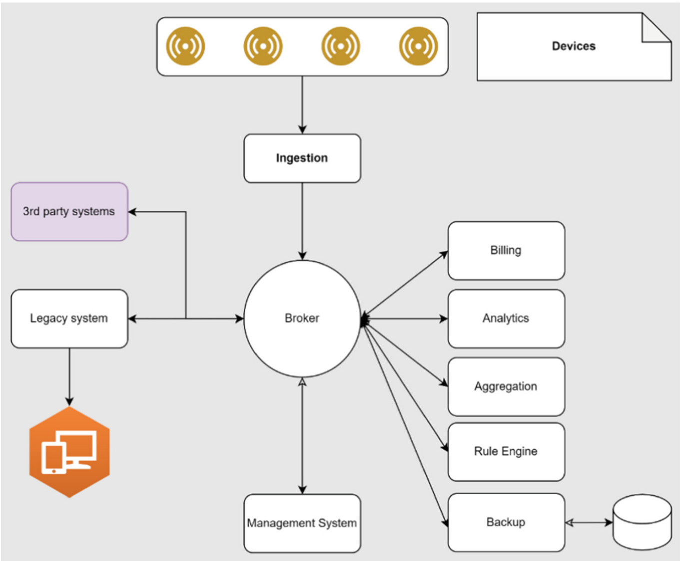

Fig. 2. The new architecture and its important components. In this architecture the ingestion layer is just a thin layer to fetch data from sources and import into the broker with no manipulation. The broker is the shared area which should be scalable enough to tolerate many concurrent accesses by different microservices to add values to data. Therefore, any entity should read from and write to this component.

In this architecture, the ingestion layer is again in charge of receiving and decoding data sent by devices. After this phase, it assigns a universal id to each received message and publishes them directly to the data broker. Ingestion publishes each stream into different buckets. Therefore, it is not a heavyweight layer anymore, and also it is not coupled with the structures of the upstream database. The broker is the heart of this architecture. The main idea is that data will be stored in different parts to enable concurrent access. It makes the architecture horizontal scalable and guarantees its interoperability by storing it in a semi-structured way. Different microservices are in charge of granting the functional structure to the incoming data streams through stream processing services. This policy can run the microservices in parallel, so the solution is scalable. The model is widely based on publish-subscribe architecture.

After preparing data, other microservices go through conducting different functionality, such as aggregation, analytics, billing, rule engine, and so on. This architecture can also accept configurations from the user and carry out related functions on a specific schedule over the data broker. Since the data access in the data broker is distributed, the model is scalable. Data aggregation might be done in memory such that only aggregated data is stored. This would reduce data storage requirements and increase the speed of further analysis. Different security policies are available in this architecture.

4 Benefits of the New Architecture

Data ownership will be ensured by a flexible access control. Moreover, in transit and at rest encryptions are enforced. Eexisting web platforms simply become subscribers to the broker. The broker itself is a dedicated process that is standards-compliant, lightweight, secure and fast. It is fit for handling all mentioned Big Data requirements. In order to ensure easier integration with other third-party platforms and benefit from the robustness of a tried and tested platform, the broker would be built around a standards-compliant technology.

All data feeders that publish to the broker would do so under well-defined topics. Using this method, all other processes can get access to the incoming data streams in simultaneously. Therefore, the horizontal scalability is guaranteed, and data is much more accessible. There would be a clear demarcation between device data and diagnostic data. New data feeds, e.g., from new technology such as the NB-IoT devices or third-party platforms, would be easily on-boarded – they would simply be another publisher that supports defined schemas. Therefore, a wide variety of data structures can be considered.

By brokering data, the current application in the future would only receive relevant data. Its job would no longer involve the export of large amounts of data. This load would be picked up by the broker and additional microservices designed for the sole purpose of doing this. Moreover, most brokers support clustering by which there would be clear options such as adding redundancy as a disaster recovery option, adding off-site back-up, and moving brokers to other servers and sites.

This architecture through the publish-subscribe model facilitates the one-to-many relationship for sharing data streams with one or more subscribers. For example, data could be shared concurrently with the existing web platform and other platforms, but a data stream could also be shared with, e.g., an aggregator. For a data broker, data would not be stored long term. This reduces the impact of any security breach. Where data is held in short-term volatile storage (e.g., memory) or non-volatile (e.g., NoSQL databases), encryption at rest should be achieved if possible.

An analytics engine can be adopted simply as another microservice. It would be a more powerful version of the data aggregator in many ways. Analytics could vary from the simple to the more complex. It could even harness artificial intelligence (AI), e.g., to spot patterns in data.

The engine (or multiple engines) would subscribe to data streams on the broker and publish back processed data or event streams. Throughout IoT systems, there are many processes and devices that need configuring. These can vary widely from configuration data for devices (e.g., for a LoRa radio transmitter, transmit frequency, and high and low thresholds for edge intelligence), to gateways (e.g., upload frequency), to alarm processes (e.g., high and low thresholds, alarm delays) and more. The broker could handle the exchange of this configuration data, with all such data passing through the broker. It would be the responsibility of the client subscribing to this feed to handle the data. For example, in the use case company Invisible Systems LTD (ISL), the cloud platform already passes configuration data down to the Ingestion layer. Instead, this would be passed to the broker, and an independent microservice would subscribe to this to conduct the related processing over the given incoming data stream.

Moving to a central broker allow a standalone, clearly defined, maintainable and scalable process to be built for each export route. Moreover, where the existing platform does not require data, it would be pushed straight out to a third party, ensuring speed, reduced data storage, and capacity for Big Data. A new approach could be envisioned around the broker, including an API process subscribing to the broker for data stream topics it is interested in; fixed-term storage is used to keep a log of recent data (e.g., for the last month, according to agreed commercial terms); this could be stored in a separate database that is specific to the process or may be stored in a NoSQL database.

5 Discussion

Some quantitative experiments have been made to compare these two architectures regarding their related technologies in Invisible Systems Ltd' data. In the new architecture, Apache Kafka is an appropriate candidate as a data broker. And a candidate database to persist data streams is influxDB. During testing, we have achieved sustained throughput in excess of 30 k messages per second with 3 Kafka broker nodes and 1 client node (AWS t3.small, co-hosted in same data center as broker). Moreover, we have achieved sustained database write speeds of 32 k data points per second (unlimited influxDB resource, writing from 1 client node, different AWS data centres). The conventional architecture writes at up to 75 devices/second (empirically equivalent to around $75 \times 5.3 = 400$ data points), from minimum 2 ingest nodes (equivalent to 200 data points per node per second). Therefore, node for node, the new architecture is $32000/200 = 160$ times faster.

The new architecture also offers additional processing, such as detecting alarms, which generate additional data points. Therefore, the load generated by the new architecture is arbitrarily larger, depending on user configuration. Based on an estimated 5 additional data points (alarms etc.) per measured data point, each node can handle $160/5 = 32$ times more load than a current data node. This yields an estimated throughput per node of $(200 \times 32)/5.3 = 1200$ devices per second. At standard transmit intervals of around 7.5 min, this equates to 540,000 devices in the field per node. The platform can scale horizontally, giving a capacity of many millions of devices. Table 2 summarizes the above findings to compare these two architectures.

Table 2. Comparison between the conventional and the proposed architectures.

Architecture	Write speed (datapoint per node)	Load number
Conventional architecture	200	5
The proposed architecture	32000	160

6 Conclusion

In this research we reviewed important non-functional requirements to design an IoT system. To this end a conventional system has been investigated, and the important requirements have been addressed. To overcome the important requirements, a new brokering-based architecture was designed and prototyped. In this research we discussed that how the new architecture could cover the non-functional requirements such as security, scalability, extensibility, resiliency and low latency. We also carried out some experiments by gathering data from ISL that showed the new architecture outperforms the existing ones.

Acknowledgement. This research has resulted from the Knowledge Transfer Partnership between the University of Salford and Invisible Systems Ltd (ISL) partially funded by Innovate UK "KTP011129".

References

Afzal, B., Umair, M., Asadullah Shah, G., Ahmed, E.: Enabling IoT platforms for social IoT applications: vision, feature mapping, and challenges. Futur. Gener. Comput. Syst. **92**, 718–731 (2019). https://doi.org/10.1016/j.future.2017.12.002

Aksu, H., Babun, L., Conti, M., Tolomei, G., Uluagac, A.S.: Advertising in the IoT era: vision and challenges. IEEE Commun. Mag. **56**(11), 138–144 (2018). https://doi.org/10.1109/MCOM.2017.1700871

Balaji, S., Nathani, K., Santhakumar, R.: IoT Technology, applications and challenges: a contemporary survey. Wireless Pers. Commun. **108**(1), 363–388 (2019). https://doi.org/10.1007/s11277-019-06407-w

Raza, U., Kharel, R.: A survey on the challenges and opportunities of the Internet of Things (IoT). In: 2017 Eleventh International Conference on Sensing Technology (ICST), pp. 1–5. https://doi.org/10.1109/ICSensT.2017.8304465

Poniszewska-Maranda, A., Kaczmarek, D.: Selected methods of artificial intelligence for Internet of Things conception, pp. 1343–1348. https://doi.org/10.15439/2015F161

Rautmare, S., Bhalerao, D.M.: MySQL and NoSQL database comparison for IoT application. In: 2016 IEEE International Conference on Advances in Computer Applications, ICACA 2016, pp. 235–238 (2016)

Shah, S.H., Yaqoob, I.: A survey: Internet of Things (IOT) technologies, applications and challenges. IEEE Smart Energy Grid Engineering (SEGE) **2016**, 381–385 (2016). https://doi.org/10.1109/SEGE.2016.7589556

Chen, S., Hui, X., Liu, D., Bo, H., Wang, H.: A Vision of IoT: applications, challenges, and opportunities with China perspective. IEEE Internet Things J. **1**(4), 349–359 (2014). https://doi.org/10.1109/JIOT.2014.2337336

Tunc, M.A., Gures, E., Shayea, I.: A Survey on IoT smart healthcare: emerging technologies, applications, challenges, and future trends. [Cs, Math]. http://arxiv.org/abs/2109.02042
Yue, Y.-G., He, P.: A comprehensive survey on the reliability of mobile wireless sensor networks: taxonomy, challenges, and future directions. Inf. Fusion **44**, 188–204 (2018). https://doi.org/10.1016/j.inffus.2018.03.005