SN Computer Science (2022) 3:493
https://doi.org/10.1007/s42979-022-01401-9

ORIGINAL RESEARCH

A Reference Architecture for IoT-Enabled Smart Buildings

Muhammad Rizwan Bashir[1] · **Asif Qumer Gill**[2] · **Ghassan Beydoun**[1]

Received: 21 August 2021 / Accepted: 5 September 2022 / Published online: 27 September 2022

Abstract
The management and analytics of big data generated from IoT sensors deployed in smart buildings pose a real challenge in today's world. Hence, there is a clear need for an IoT focused Integrated Big Data Management and Analytics framework to enable the near real-time autonomous control and management of smart buildings. The focus of this paper is on the development and evaluation of the reference architecture required to support such a framework. The applicability of the reference architecture is evaluated by taking into account various example scenarios for a smart building involving the management and analysis of near real-time IoT data from 1000 sensors. The results demonstrate that the reference architecture can guide the complex integration and orchestration of real-time IoT data management, analytics, and autonomous control of smart buildings, and that the architecture can be scaled up to address challenges for other smart environments.

Keywords Internet of things · Smart buildings · Data analytics · Big data management

Introduction

An increasing number of Internet of Things (IoT) initiatives have been proposed in recent times to improve the quality of human life. Those initiatives pose real-time challenges which have been the focus of many researchers and practitioners in recent times [1–5]. Indeed, IoT and big data sources can be found in a number of applications, e.g., smart homes [6, 7], smart buildings [8, 9], smart grids [10], transportation [11], healthcare [12], disaster management [13], financial sector [14], retail management [15], and smart cities [16, 17]. IoT sensors can be deployed in a smart building environment to continuously monitor various environmental parameters, including smoke, parking lot usage, user comfort, energy consumption, waste management, and many others. The aim of this paper is to facilitate the use of analytics and dealing with the concomitant large data sets in smart buildings for the effective control and management of smart building.

Within a smart building, the number of sensors could range from few hundreds to thousands. Big data analytics and machine learning techniques can only be effective, if the data from sensors are effectively managed and are made available and ready for real-time analytics. This real-time 'Big Data' needs to be extracted and ingested into a centralized location from where it can be extracted, cleaned, transformed, analyzed, and visualized on-demand or in real time [18] to obtain useful insights, to make effective decisions, and eventually trigger alerts and actuate various controls in a smart building.

Real-time strict definition is that "an upper bound" on the response time actually exists [19]. We use the term 'near real-time' in this work as there is an insignificant data processing delay involved when analyzing IoT sensor data [20]. Strictly speaking near real-time can be defined as "in more than 95% of cases, an upper bound on the response time of 1 s will not be exceeded". In the context of smart buildings, some of the challenges include responding to emergency situations in real time and the possibility of autonomously eliminating or reducing it.

To deal with the challenges of real-time big data management and analytics in the smart building context, a coherent framework which incorporates a metamodel and a reference architecture is needed. This research is aimed at bridging

✉ Muhammad Rizwan Bashir
rizwan.bashir@student.uts.edu.au

Asif Qumer Gill
asif.gill@uts.edu.au

Ghassan Beydoun
ghassan.beydoun@uts.edu.au

[1] School of Information Systems and Modelling, University of Technology Sydney, Ultimo, NSW 2007, Australia

[2] School of Computer Science, University of Technology Sydney, Ultimo, NSW 2007, Australia

this gap in the literature. The metamodel provides the list of essential elements in a smart building ecosystem and how these elements interact with each other. The reference architecture on the other hand provides an end-to-end blueprint to enable real-time management and analytics of huge amounts of IoT data coming from various IoT sensors. It also intends to provide autonomous near real-time control of smart buildings by analyzing, monitoring, and controlling various facilities within the smart buildings. The reference architecture and the metamodel are linked to each other through five contextual elements.

IoT sensors deployed inside a smart building gather useful information like residents' occupancy, oxygen levels, luminosity levels, etc., which help manage and secure the smart building more efficiently. IoT is the core building block for today's smart buildings, and it enables artificial intelligence and big data analytics for smart building operations. With IoT, data from various buildings can be observed, gathered, and analyzed, and the IoT sensors can be updated with the latest software from anywhere across the globe. This paper is an extension to our work presented in [21] and [22]. In [21], we presented the idea of an IBDMA (Integrated Big Data Management and Analytics) framework, which comprises of a metamodel and reference architecture. The proposed IBDMA framework is aimed at addressing two issues: (1) how to effectively manage and analyze data generated by IoT sensors deployed inside smart buildings, and (2) how to holistically identify all the elements and the relationship between these elements to effectively manage and analyze, i.e., data in IoT-enabled smart buildings. The first issue is addressed by this paper, while the second issue has been addressed in [21]. There is no coherent framework which provides a metamodel and a reference architecture to address the issues outlined above. The existing frameworks in the literature either provide a reference architecture or a metamodel, but there is no framework in the literature which provides a coherent view of the two. The aim of the IBDMA framework is to enable developers design the smart building by providing a comprehensive list of components required in a smart building by utilizing the IBDMA metamodel. The metamodel will also enable to convert an existing building into the smart building. The IBDMA reference architecture enables researchers and practitioners to manage and analyze IoT data in a smart building efficiently. The metamodel work has been presented in [21]. In this paper, we present the second component of the IBDMA framework: the IBDMA reference architecture. The aim of the IBDMA reference architecture is to enable big data management and analytics within smart buildings while autonomously monitoring and controlling various facilities within the smart building.

The applicability of the IBDMA reference architecture is demonstrated using it for smart building experimental scenarios to monitor and control oxygen concentration levels, luminosity levels, smoke levels, parking lot spaces, and waste management with a view to improve residents' safety, health, and comfort. The IoT data are presented using multiple data visualization tools. We use ARIMA (Auto Regressive Integrated Moving Average) [23] model to forecast values of IoT sensors, and suggest that the reference architecture can be employed and extended in the machine learning domain for data scientists and machine learning practitioners. However, the choice of ARIMA model, its fine tuning, and evaluation are beyond the scope of the paper.

The rest of the paper is organized as follows: "Research Background and Related Work" presents the research background and related work. "Research Method" discusses the method used in this research. "The IBDMA Framework" presents the conceptual level IBDMA framework. "Reference Architecture Development Process" discusses the development and iterative process of the IBDMA reference architecture. "Reference Architecture Implementation" provides implementation details of the architecture. "Framework Evaluation Results" presents the evaluation details and the evaluation results. "Conclusions and Future Work" concludes and discusses future research directions.

Research Background and Related Work

Research Background

This section discusses the research background related to the development of the IBDMA framework. It elucidates the background knowledge required to understand the IBDMA framework. Since this research focuses on a reference architecture for IoT-enabled smart buildings, there are two important concepts to understand, which are explained below:

Internet of Things

The Internet of Things (IoT) refers to a network of interconnected devices with the ability to exchange information via Internet [24]. IoT has become an increasingly popular topic of interest both in academia and industry. This includes everything from wearable devices, mobile phones, heart monitor implants, or any other type of sensors (oxygen, luminosity, garbage detection, etc.), which have the ability to transfer data over the Internet. IoT applications can be found in many domains ranging from precision agriculture, smart cities, smart buildings, smart grids, healthcare, transportation, and many more.

IoT has seen a tremendous growth in the past couple of years and this growth rate is expected to increase in the upcoming years. It holds a lot of promises [25]. According to Ericsson [26], the number of interconnected objects is expected to rise above 50 billion mark by 2020. The IoT

devices range from sensors used inside homes for home automation [27], sensors deployed in smart buildings [28], sensors installed in vehicles [29], sensors inside a warehouse [30], sensors integrated inside wearable devices [31], and many others [32, 33]. According to another report, considering this tremendous pool of sensing devices, it is anticipated that the number of IoT devices will reach trillions of number [34] in the upcoming years. However, such an increase in the number of IoT devices will also increase the amount of data generated from these sensors. This increasing number of sensors and huge amount of data raise new challenges and concerns for data management and analytics practitioners, scientists, researchers, and data architects. An IoT system comprises of four high-level layers or blocks, as shown in Fig. 1. Perception layer consists of sensors and actuators. Sensors gather data from the environment and actuators are activated based on the data gathered from the sensors. Network layer enables sensors to connect to internet. Middleware later consists of data storage and computing engines. The application layer consists of applications including dashboards and reports.

In this research, we use five different types of sensors which monitor the environment, sense various parameters depending on the type of the sensor and generate data at a specified frequency. The generated data through TCP/IP (Network Layer) are stored into a central location (Middleware Layer). The data are analyzed (Middleware Layer) and are reported using visualizations (Application Layer). The next section explains the data generated by IoT sensors in more detail.

Real-Time Big Data

The concept of real-time (or near real-time) data management and analytics (within the IoT paradigm) refers to capturing, storing, and analyzing the data streams as soon as they are received from the IoT sensors. IoT sensors deployed in the smart building generate a lot of data at a high velocity, comprising big data. Big data management refers to sourcing, storage, and distribution of data. Analytics refers to finding patterns and valuable insights using various analytical techniques and algorithms [36]. In today's fast-moving digital age, businesses want to stay ahead of their competitors by focusing on the immediate implications of managing and analyzing real-time data. Real-time data management and analytics can be categorized into two types—On-Demand and Continuous [37], which are distinguished by the reactive and proactive approaches [38]—or pull and push [39]. For instance, On-Demand Real-Time Analytics is reactive. It waits for user to initiate a query and it delivers the results. Continuous Real-Time Analytics, on the other hand, is more proactive and keeps on delivering the analytics results to the users in real-time. Both above-mentioned types of real-time data analytics have their own use cases and can be used to provide valuable information and insights to a business to make effective decisions. In this research, we focus on both data analytics techniques.

The duology of IoT and, near real-time big data management and analytics is complex in nature. There is a lack of integrated and coherent framework. This research integrates a metamodel and a reference architecture to address the real-time big data management and analytics challenges in smart buildings. This research aims to address this important challenge by proposing the IBDMA framework's reference architecture using the well-known DSR [40] method. The IBDMA framework has five key contextual elements: people, process, technology, information, and facility. The metamodel and the reference architecture are connected through these five key elements.

Related Work

Increasing interest in the areas of IoT, big data management, and big data analytics in recent years has been observed both in academia and industry. In [41], authors put forward different big data analytics techniques and specifically discuss Apache Spark in the context of smart grid big data. In [42], the authors propose a real-time semantic annotation reference architecture for Smart City IoT streaming applications. The work presented in this paper provides a foundation for the development of a comprehensive framework that could be useful in improving the performance capability of a smart city. In another recent study [36], an IoT-based Wireless Sensing and Monitoring platform has been proposed

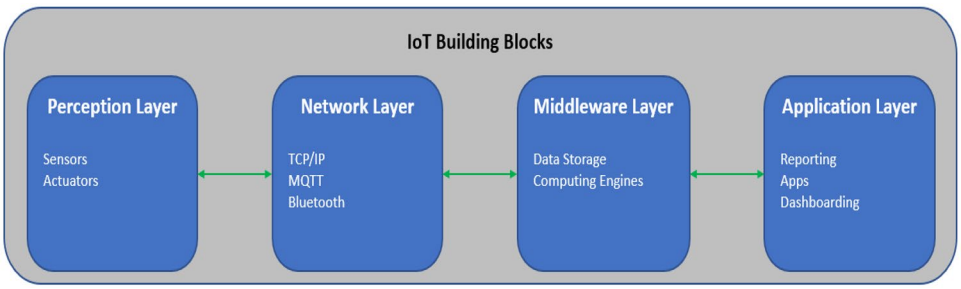

Fig. 1 IoT architecture [35]

to detect environment conditions in the context of building automation. The work presented in this research does not, however, discuss the real-time data analytics for the IoT-enabled smart environments.

In [43], the authors discuss the advantages of sensing and analyzing big data from various sensors in a smart city. This paper provides a conceptual overview and the advantages of applying big data techniques over IoT data coming from sensors deployed in the smart city. It also highlights the difference between static and mobile data sources and proposes the best data extraction techniques for both types of data sources. Similarly, in [44], the authors discuss different big data analytics techniques suitable in a smart city scenario. This paper also discusses some of the major challenges in the data analytics process for smart cities data. In [45], the authors survey IoT, Cloud Computing, Big Data, and Sensor technologies with the aim to find their common operations and combine them. The authors then propose new methods to collect and manage sensors' data in a smart building. In [46], the authors present a distributed system for storing and processing building data. Based on the big data technologies, the platform enables new potentials in terms of data analytics for smart buildings' applications. These papers, however, do not provide an insight or concrete guidelines and implementation architecture, as discussed in this paper, about which and how different components can be integrated and implemented to design real-time data management and analytics architecture and solutions.

In [47], the authors propose a social media data analytics platform that uses tweeter posts to improve the smart city experience for the residents of the city. It aims to improve the residents experience by analyzing real-time Twitter posts. The overall results suggest that this platform can help improve the effective management of the smart city. This paper, however, only takes into account residents' sentiments and twitter posts without taking into account IoT sensor data and actuation of the controls. An IoT-based system has been implemented in [48], this paper focuses on obtaining real-time sensor data from IoT sensors deployed in a smart city and performs the real-time analytics on it. A practical demonstration has been presented in the paper using Hadoop ecosystem. This paper, however, does not address the smart city control and end-to-end data management scenario.

In [49], the authors present a scalable architecture for ingesting and analyzing IoT data called the hut architecture. It utilizes historical data analysis to provide context for real-time analysis. The applicability of the architecture is demonstrated using two real-world smart city scenarios in transportation and energy management.

In [50], the authors present an initial version of Big Data analytical framework for Internet of Things and Smart City application. This work demonstrates how such a framework can be used by presenting a case study in the smart grid domain. However, this framework is a high-level and initial version addressing some of the volume and velocity challenges. The implementation details around the use of tools and the data ingestion pipelines are not made clear. Moreover, the results obtained from the analytics have not been used to autonomously control the smart city or smart grids based on the received data.

In [51], the authors presented a Big Data architecture for the Smart Supply Chains fields. Data were ingested into Hadoop and Machine Learning models were used to address data-related challenges in Supply Chains. However, this paper lacks details in explaining the significance of the work, and how it can be applied in real-life scenarios. It presents only the architecture of an IoT pipeline which is missing the near real-time visualizations as presented in our work. And more importantly lacks the framework we have developed with five elements: 1—People, 2—Process, 3—Technology, 4—Information, and 5—Facility, and how these elements are related to the underlying Big Data Management architecture and to Smart Buildings.

In [52], the authors presented the use of machine intelligence and data analytics algorithms on data acquired from the sensing networks' integral to smart city applications. However, the paper lacks the conceptual framework and lacks the implementation details of the proposed architecture. However, it presents a very high-level architecture which lacks any conceptual framework and implementation details of how the authors performed their experiments and evaluation. It is a very generic paper listing some of the machine learning and data analytics challenges in smart cities.

In [53], the authors explore the IoT issues in smart buildings and compare two network protocols used for IoT devices to improve energy efficiency in smart buildings. However, this paper lacks a focus on the big data management and analytics of IoT data in smart buildings. It also lacks the discussion on the control and management of various controls and facilities within the smart building.

In [54], the authors present a technique for the facial recognition in smart cities. This paper lacks a reference architecture for smart buildings. Kuma et al. [55] discuss IoT applications and challenges in various domains. It lacks big data analytics focus and highlights that it is a key challenge. Kuma et al. [56] present IoT-based fog computing model. It lacks a focus on big data management and analytics for IoT data. It also does not focus on smart building domain.

In [57], the authors focus on the service-oriented architecture and the networking layer. It does not focus on big data management and analytics, near-real time visualization, and the autonomous control of smart buildings. In [58], the authors focus primarily on the networking layer, identifying which protocols are available and a comparison of those communication protocols. It identifies the future challenges,

but it does not mention the data management and analytics in smart buildings. It is a survey paper which does not specify any framework or reference architecture like the IBDMA. It also lacks discussion about the real-time analytics and control of the smart buildings. In [59], the authors suggest a big data mining IoT system. It does not focus on autonomous control of the facilities in general and smart building controls in particular. It mentions about the generic data gathering systems, but details about the implementation and evaluation of the suggested system are missing. In [60], Gubbi et al. present the architectural elements in IoT paradigm, but lack a focus on (1) data management and analytics, (2) near real-time analysis, (3) near real-time visualization, and (4) near real-time control of facilities within smart building. It also lacks the integration of the architecture with a metamodel which the IBDMA framework provides.

In [24], the authors present a high-level IoT architecture. It does not mention any details about big data management and analytics, near-real time visualization, and the autonomous control of smart buildings. In [61], the authors present high-level IoT architecture and challenges faced in the IoT domain. It lacks autonomous control of the facilities in general and smart building controls in particular. It is a survey paper which does not specify any framework or reference architecture like the IBDMA. It also lacks discussion about the real-time analytics and control of the smart buildings. In [21], the IBDMA framework's second component, i.e., the metamodel is presented. The metamodel presents the key elements and the relationship between these elements that are required in the big data management and analytics ecosystem for smart buildings. However, the paper does not present the reference architecture, and hence, this research focuses on the second component of the framework, i.e., the reference architecture.

Table 1 summarizes the research gaps and the corresponding studies where we observed the research gap.

Based on the literature review and Table 1, it is quite evident the literatures lacks focus on real-time big data management and analytics, integration of a reference architecture and metamodel, and real-life validation scenarios in the smart buildings context; and hence, there is an urgent need for a vendor independent practical research-based integrated comprehensive framework for IoT real-time big data management and analytics. In this research, we presented the IBDMA framework, which is an attempt to fill the research gap. This sets a foundation for more studies in this important area of research.

Research Method

This research adopts the DSR approach [62]. DSR proposes a practical research approach supporting the creation of artifacts to solve real-life problems [63]. DSR encompasses the formation of new information through the design of novel artifacts. It involves the performance analysis of such artifacts to understand and improves the behavioral of aspects of IS (Information Systems) [64]. These artifacts may include algorithms, methodologies of system design, and human/computer interfaces. DSR researchers can be found in various domains, e.g., Engineering, Information Systems and Computer Sciences. In [65], DSR activities are described for the IS discipline using a conceptual framework. We adopt the guidelines from [63, 65] in conducting this research, as shown in Table 2. The rationale behind using the DSR approach for this research is that; first, the research involves incremental development of the reference architecture; second, the DSR focuses on solving real-world problems, and for this research, we address the challenge of big data management and analytics by developing a reference architecture; thirdly, DSR tries to address the gap between theory and practice, and for this research, we are trying to address the gap in the areas of BDMA for IoT-enabled smart buildings.

For assessing the quality of our DSR, we follow the checklist questionnaire also from [31]. The checklist is presented in Table 3.

We adopt the 'Three Cycle' DSR framework from [66] for conducting our research. The Relevance Cycle links the contextual environment with the DSR activities. The Rigor Cycle bridges the DSR activities with the knowledge base of scientific theories and methods, that inform the research. The centrally located Design Cycle continuously iterates between the development and evaluation elements of the DSR. The three cycles mentioned above must exist in DSR project and must be distinguished clearly from each other. Following these research cycle and the checklist questions of Table 3, we first identified the research question. Then, we defined the artifacts and the design processes that would be used to build those artifacts. The literature was reviewed to

Table 1 Research gap

Research gap	Studies where gap was observed
Lack of real-time big data management and analytics in smart buildings context	[24, 36, 41–44, 46–61]
Lack of integration between reference architecture and metamodel in smart building context	[24, 45, 60]
Lack of real-life validation of reference architecture in smart building context	[52, 59]

determine if the knowledge base provides support to the artifact design. The artifacts were then designed, and an evaluation method was proposed to test the designed artifacts. Finally, the research is communicated in the form of publication. Figure 2 represents these three research cycles and each checklist question from Table 3 mapped onto appropriate phases of the three-cycle DSR approach.

There are five main steps involved in the DSR, as shown in Fig. 3. In the first step, the problem is identified by finding the research gap. We performed literature review to identify the research gap, and eventually helped us identify the problem. We then proposed the design of an IBDMA framework which consists of a metamodel and a reference architecture to bridge the research gap. The IBDMA framework components (metamodel and reference architecture) were designed and developed to address big data management and analytics challenges in smart buildings by providing a holistic view of all the elements required in a smart building ecosystem. The IBDMA framework components were then evaluated, and then, the evaluation results are presented as outcomes of

Table 2 DSR guidelines [35]

#	Guideline	Description
1	Design as an artifact	DSR must produce a reasonable artifact in the form of a concept, a model, a process, or an instantiation
2	Problem relevance	The goal of DSR is the design and analysis of technical solutions to address vital real-life issues
3	Design evaluation	The usefulness and effectiveness of a design artifact must be tested comprehensively using well-defined evaluation standards
4	Research contributions	DSR must contribute effectively and clearly in the areas of the design artifact, design methodologies, and design foundations
5	Research rigor	DSR depend on the application of rigorous methods in both the development and evaluation of the design artifact
6	Design as a search process	Finding an effective design artifact relies on utilizing available resources to achieve the objectives while satisfying rules in the problem environment
7	Communication of research	The findings of the DSR must be communicated effectively to both technical and non-technical audiences

Table 3 Checklist to assess design science research [65]

#	Questions
1	What are the design requirements or research question?
2	What is the artifact and how is it denoted?
3	How to build the artifact from the design processes?
4	How the knowledge base lays the foundation of generating the artifacts?
5	During the design cycles, what evaluations are done? Are there any design improvements that are identified during each design cycle?
6	How is the newly generated artifact introduced and is evaluated in the field? What parameters are used to evaluate the usefulness of the artifact over existing artifacts?
7	What new information is added to the existing literature and knowledge base?
8	Does the newly generated information satisfy the research question satisfactorily?

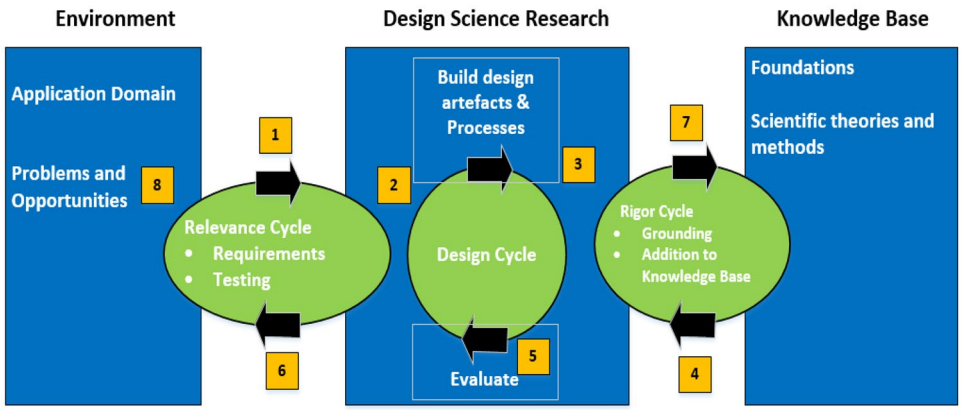

Fig. 2 Checklist questions mapped to three DSR cycles [65]

Fig. 3 DSR steps

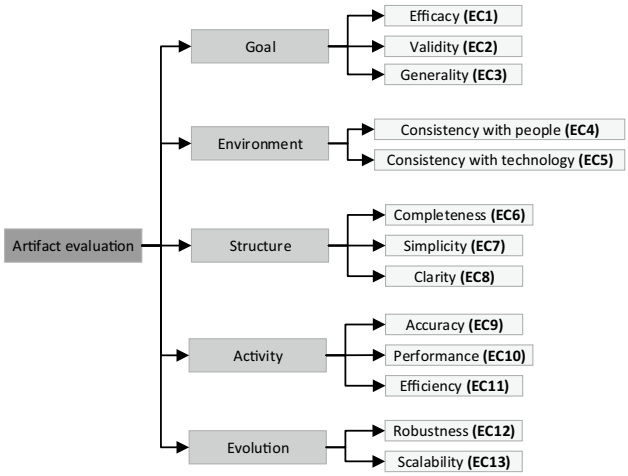

Fig. 4 ECs for IBDMA evaluation (adopted from [67])

the research. As mentioned earlier, the metamodel development and evaluation has been published in [21]. This paper focuses on the IBDMA reference architecture, which is the second component of the IBDMA framework. Following the DSR approach, we went through five iterations of the IBDMA reference architecture before coming up with a final reference architecture as previous iterations did not satisfy the evaluation criteria. The details about all five iterations, the development, and evaluation processes are explained in the next sections.

The framework we have developed has been evaluated against the EC (evaluation criteria), as shown in Fig. 4. In [67], the authors present artifact evaluation criteria in design science research. The 13 evaluation criteria we shortlisted (out of 20 from [67]) have been adopted from [67] based on their relevance to our research, as some of the evaluation criteria were not applicable to our research or were not possible to be evaluated concretely (such as style, homomorphism, level of detail, etc.). For this research, we chose 13 ECs, i.e., EC1, EC2, EC3, …., EC13 for evaluating the IBDMA framework and reference architecture. The details of these ECs are presented in Fig. 4. These ECs were evaluated against our research objectives of:

- Ability to have both batch and streaming analytics.
- Ability to do near real-time analytics and visualization.
- Ability to autonomously control facilities within smart building.
- Ability to provide building management and relevant authorities the alerts in near real time.
- Ability to scale up for any building size and for other smart environments.
- Ability to provide a comprehensive framework comprising of a reference architecture and metamodel.
- Ability to validate the proposed design using real-life scenarios.

The evaluation results are presented in "Framework Evaluation Results" in Table 11.

Table 4 provides details of the research objectives, and how they are lined to the research gaps and research aims.

The IBDMA Framework

The proposed IBDMA framework, as shown in Fig. 5, consists of two components. This paper specifically covers the reference architecture component. The metamodel part has been developed earlier and is presented in [21]. The reference architecture is encircled in Fig. 5 to demonstrate that this is the focus of the research. With the reference architecture, IBDMA framework will assist professionals and researchers working in the big data and IoT domains in the smart building context. Figure 5 represents that the context of the research is limited to smart building, and the IBDMA framework (applicable with the smart building) has two components: (1) Metamodel and (2) Reference architecture. The metamodel and the reference architecture are linked to each other through the contextual elements.

The IBDMA framework as adopted from [68–71] has five main contextual elements, as shown in Fig. 6.

The IBDMA framework will help building developers, and IoT and big data professionals to have a holistic view of which elements do they need to deploy in the smart building while designing the smart building or converting an existing building into smart building. The metamodel helps in identifying those elements and the relationship between

Table 4 Research objectives linked to research gaps and research aims

Research objectives	Research gaps	Research aims
Ability to have both batch and streaming analytics	Lack of real-time big data management and analytics in smart buildings context	How to effectively manage and analyze data generated by IoT sensors deployed inside smart buildings
Ability to do near real-time analytics and visualization	Lack of real-time big data management and analytics in smart buildings context	How to effectively manage and analyze data generated by IoT sensors deployed inside smart buildings
Ability to autonomously control facilities within smart building	Lack of real-time big data management and analytics in smart buildings context	How to effectively manage and analyze data generated by IoT sensors deployed inside smart buildings
Ability to provide building management and relevant authorities the alerts in near-real time	Lack of real-time big data management and analytics in smart buildings context	How to effectively manage and analyze data generated by IoT sensors deployed inside smart buildings
Ability to scale up for any building size and for other smart environments	Lack of real-time big data management and analytics in smart buildings context	How to effectively manage and analyze data generated by IoT sensors deployed inside smart buildings
Ability to provide a comprehensive framework comprising of a reference architecture and metamodel	Lack of integration between reference architecture and metamodel in smart building context	How to effectively manage and analyze data generated by IoT sensors deployed inside smart buildings How to holistically identify all the elements and the relationship between these elements to effectively manage and analyze, i.e., data in IoT-enabled smart buildings
Ability to validate the proposed design using real-life scenarios	Lack of real-life validation of reference architecture in smart building context	How to effectively manage and analyze data generated by IoT sensors deployed inside smart buildings

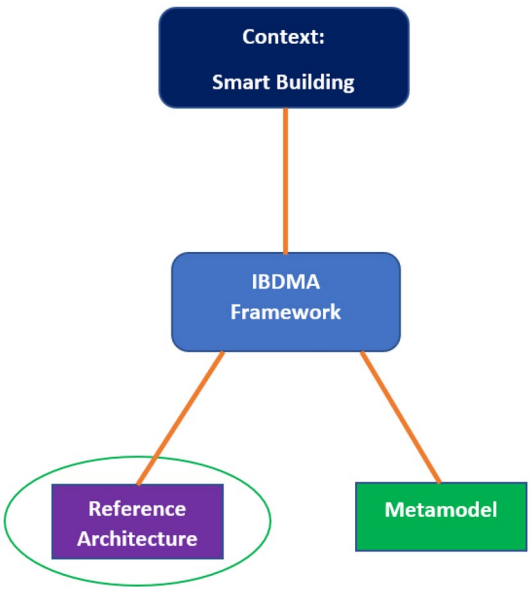

Fig. 5 IBDMA framework and scope of the paper

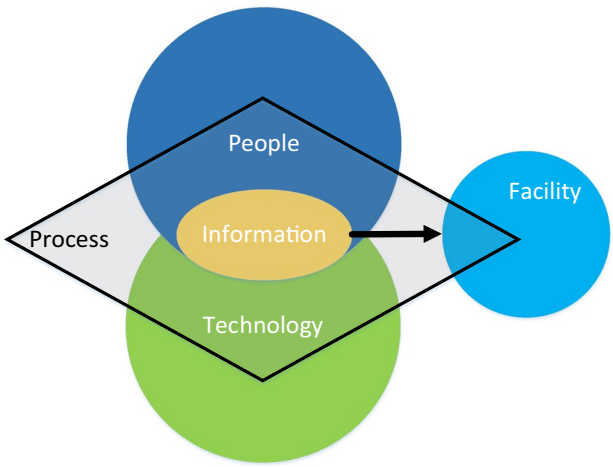

Fig. 6 IBDMA framework—contextual elements

those elements, while the reference architecture provides the ability to link those elements to the physical design of the end-to-end data management, analysis, and process flow.

The reference architecture (which is the focus of this paper) is novel in that it provides a scalable architecture which focuses on big data management and analytics for smart buildings and can be extended to other smart environments (smart homes, smart grids, and smart cities). The reference architecture also provides an ability to mitigate the risks and improve residents' experience in smart building by providing the ability to autonomously control various facilities within the building.

The integration of metamodel and reference architecture is important to provide a holistic view of the IoT-enabled

smart building ecosystem for big data management and analytics of IoT data. The integration of the metamodel and the reference architecture not only assists in developing new smart buildings but also provides the ability to transform an existing building into a smart building by enabling big data management and analytics for IoT data. In the initial phase of transformation, the metamodel will be used by the building developers, architects, and administrators along with IoT experts to identify which elements do they need and to understand the relationship between these elements. Once this has been achieved, the big data developers and architects can utilize the reference architecture to implement the big data management and analytics processes in the smart building ecosystem.

As shown in Fig. 6, 'People' element is the core IBDMA element. This includes the 'residents', 'policy-makers', as well as the developers of the smart buildings. The 'policy-makers' make policies which enable and govern the smart buildings ecosystems. The 'developers' develop the smart buildings adhering to policies compiled by the 'policy-makers'. The 'residents' may include students, staff, home owners, shop owners, etc. These are beneficiaries of the smart building ecosystem. The 'developers' and 'policy-makers' make policies which help identify 'Process' element of IBDMA. The 'Process' element encompasses all processes which are required for the effective management and analysis of the smart building data. The 'technology' element consists of the technology stack that supports the processes as defined by the 'process' element of the framework. The overlap of these elements results in useful information which makes up the fourth element of IBDMA known as 'information'. The information is then autonomously used to control various facilities within smart buildings, which fall under the 'facility' element of IBDMA. The 'process' element links all other elements, as shown in Fig. 6. The way these different elements are linked and interacted is explained in the upcoming sub-sections.

People

'People' element which is the first element IBDMA. It includes policy-makers, developers, and building residents, as shown in Fig. 7. These can be broken down into two groups, one consisting of policy-makers and developers, and another consisting of building residents.

Policy-Makers and Developers

The policy-makers define policies which govern the policies of the building. These policies define the key requirements from the stakeholders and help propose the optimum solution to meet the expectations of the stakeholders and residents.

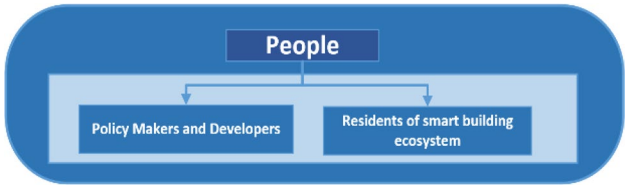

Fig. 7 The first element of IBDMA—people

The developers include the building developers who develop the smart building in line with the policies and regulations. Their role is to ensure the safety, security, and comfort of the residents of the building.

Residents

The residents of the building may include students, staff, tenants, homeowners, shopkeeper, etc. depending on the nature of the smart building. They are the users or beneficiaries of the smart building ecosystem.

IBDMA proposes that based on the policies defined by the policy-makers, the "processes" are defined which are required for the effective execution of these policies. These "processes" help identify "technology" stack required for the effective execution of these "processes". This includes tools and software applications required for the execution of the "processes", e.g., Microsoft Power BI [72] and Tableau [73] for data visualization, Apache Flume [74] for data ingestion, Apache Spark [75] for data analysis, etc. The applicability and usability of each of these tools and the process elements is explained in detail in the next sections.

Policy-Makers and Developers (people) clearly articulate the requirements for the smart building. Processes are then identified and executed. This includes ingesting IoT data, storing it, and analyzing it. Hence, 'process' element is the second element that follows when applying the framework.

Process

"Process" is the second element of IBDMA framework. It performs a key role in defining the strategy for the implementation of IBDMA framework. Processes outline the operations and how various operations should be integrated for a concise and effective solution. Hence, to have an effective solution, processes defined in IBDMA should be transparent and streamlined.

The goals, requirements, and policies defined by the "people" serve as input to defining the processes and form the basis for choosing and implementing the "processes". Since this research focuses on ingesting, managing, and analyzing data generated by IoT sensors deployed in smart buildings, the IBDMA framework proposes the following

processes to by implemented which include: monitoring of the smart building environment, sourcing of IoT data, ingestion of data, storing of data at a centralized location, analyzing near-real time, decision-making, visualizing of near real time, and autonomously controlling various smart facilities in the smart building in near real time as presented in Fig. 8. The processes depicted in Fig. 8 represent the data flow and hence are sequential (monitoring, data sourcing, ingestion, analysis, decision-making, and actuation). There may be several facilities that could be controlled autonomously within the smart building using IBDMA framework. However, to have a realistic scope for this research, we consider five facilities which include managing oxygen levels, luminosity levels, garbage, parking, and fire.

Monitoring In all IoT systems, the first process in implementing a big data management and analytics infrastructure is the 'monitoring' of the environment in which the IoT sensors are deployed, which for this research is the smart building. There are various types of IoT sensors available these days that could be deployed to monitor various parameters and attributes of the smart buildings depending on the use cases and requirements of the residents and stakeholders.

Data Sourcing On monitoring the environment in which they are deployed, these sensors generate data. The output from these sensors could be binary or continuous depending on the nature and type of the IoT sensors.

Data Ingestion The data generated from these sensors are then ingested into a centralized repository using an ingestion pipeline.

Data Storage The centralized repository is where the data are stored for cleaning, manipulation, and further processing. Once the data are at a centralized location, it is made ready for the analysis.

Data Analytics The nature of data analysis is dependent on a particular use case or the requirement of the stakeholders. The analysis process is the process which enables us to obtain useful insights about the smart buildings.

Decision-Making The output of the analysis helps us in decision-making to manage and control the smart building. The decision-making process involves making decisions on whether to activate or deactivate controls in the smart building based on the data received from the IoT sensors. Decision-making processes for this research include: (1) whether an HVAC system needs to be turned ON or OFF based on the oxygen levels in the building, (2) deciding whether the lights need be to turned ON or OFF based on the luminosity levels in the building, (3) deciding whether a fire extinguisher and a fire alarm need to be triggered if the smoke levels are above a given threshold, (4) deciding whether the garbage bins need to be emptied or not if the bins are filled above a threshold level, and (5) whether the parking lot is full and the incoming vehicles can be directed to another parking lot.

Actuation/Control Based on the data analytics results and the decision-making process, the smart building controls are actuated and controlled in an autonomous manner, so the building can be managed in an effective manner.

All these different processes from monitoring to ingestion, from storage to analysis, and from decision-making to autonomous control of the smart building fall under the 'Process' element of the IBDMA which performs the core function of integrating all the elements of the IBDMA, as shown in Fig. 6. A more detailed implementation is presented in "Reference Architecture Implementation" where it will become more evident on how different elements of the IBDMA interact with each other to have an effective solution.

Once the processes are defined following the requirements compiled by the 'people', the implementation of the 'process' requires 'technology' stack. Choosing the right tools and software packages is imperative to the success of an effective solution. Within IBDMA, these tools, technologies, and software packages fall under the 'technology'

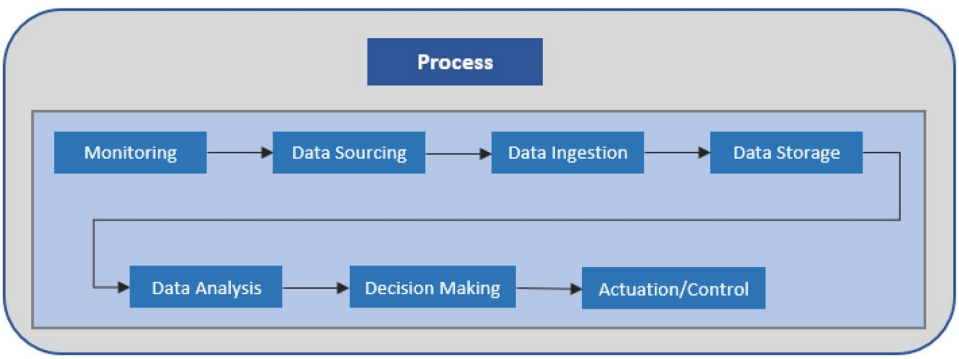

Fig. 8 The second element of IBDMA—process

element of IBDMA, and hence, it is the third element to be discussed.

Technology

'Technology' is the third element of the framework. It has a pivotal role in the effective implementation of big data management infrastructure and strategy. Hence, choosing the right technology stack is imperative. Technology includes tools and software packages deployed for effectively designing and deploying of IBDMA, as shown in Fig. 9. In general, the technology stack would include data ingestion tool, data storage tool, data visualization tool, and near real-time data analysis tool. However, for implementation and evaluation purposes, details about specific tools are provided that were used.

IoT Devices/IoT Application For this research, in the initial two iterations, we use physical IoT devices for the data analytics and building control. The details of these iterations and the use of physical IoT devices are presented in IBDMA reference architecture development process section (see iteration 1 and iteration 2 details). However, to have a scalable reference architecture which can take into account hundreds or thousands of sensors, we implement a virtual IoT sensor-based application in Python programming language and used PyCharm (a Python IDE) for the development. This application simulated data generation from IoT sensors deployed in smart buildings. Iterations 3, 4, and 5 present details of the IoT application.

HDFS The data generated by these sensors are stored in HDFS (Hadoop Distributed File System) which is a high-performance distributed file system and provides reliable data access to Hadoop clusters.

Apache Flume The data generated by the sensors are ingested into HDFS using data pipelines that are developed using Apache Flume which is a reliable data collection, aggregation, and transportation tool to ingest huge amounts of batched and streaming data, including logs, IoT data, financial data, etc., and move it to a centralized location. Flume is a fault tolerant tool which provides failover and recovery mechanisms and uses a simple extensible data model that allows for online analytical application.

Apache Spark Apache spark is used to analyze the data generated by the IoT sensors. Apache spark is an in-memory data processing engine which provides fast data analysis and processing capabilities for various streaming and batched applications. Its architecture is based on Resilient Distributed Dataset (RDD) which provides fault tolerant way of maintaining multiset of data items distributed over a cluster of machines. For this research, we use Python to write Spark code for data analysis. The analysis helps in decision-making and in turn enables the system to control and maintain the smart building and its various facilities autonomously. The aim of this autonomous is to make the smart building comfortable and secure building residents.

Power BI For visualization, Microsoft Power BI is used. Power BI comes with a built-in connector to connect to HDFS and enables to write code in R and Python to perform predictive analytics within its environment. Hence, it was a natural choice for the data visualization tool for this research. However, like any other tool, Power BI has some limitations, and it is hard to build a near-real-time dashboard in Power BI.

Elasticsearch and Kibana Since for this research, we are working with IoT sensors data, we needed to have the capability to present the IoT sensors data in near-real-time dashboard to have a greater insight into monitoring the smart building environment, so any alarms can be addressed in near-real time. To have this near-real-time visualization capability, we choose Elasticsearch [76] and Kibana [77]. Elasticsearch is an open-source tool built on Apache Lucene [78] and provides distributed search and analytics engine. New incoming data are stored as documents in Elasticsearch using either API or an ingestion tool such as Logstash [79]. This receives and stores incoming data and augments a searchable reference to the data (document) in the cluster's index. These documents can then be searched and retrieved using the Elasticsearch API. For the data visualization, Elasticsearch provides an open-source data visualization plug-in called Kibana which provides near-real time visualization

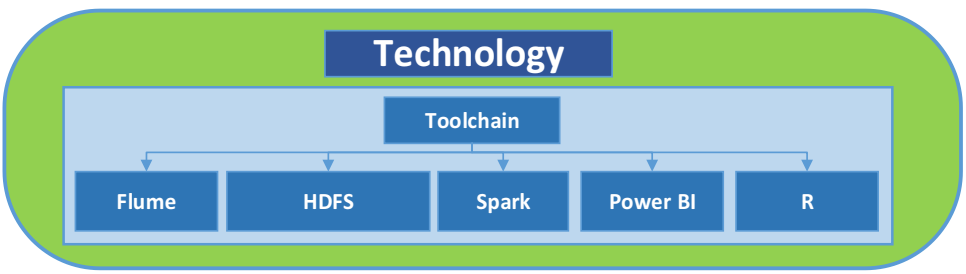

Fig. 9 The third element of IBDMA—technology

capabilities in accessing documents on an Elasticsearch cluster.

Information

'Information', which is the fourth element of IBDMA, originates from the overlap of the first three elements shown in Fig. 6. As discussed in the previous sections, 'people' outline the policies and requirements for the smart buildings. Based on these policies, processes are identified to define the 'technology' stack for their implementation. Information is generated from the data generated by the IoT sensors when the processes and the technology infrastructure are deployed successfully. There could be various forms and types of information that could be obtained from the data generated by the IoT sensors which enable us to control various parameters and aspects of the smart buildings for improved residents' comfort and safety.

Information for Building Control

There could be numerous facilities within the smart building that could be autonomously controlled as a result of the information that is generated which may include: smart lighting based on the luminosity levels, smart parking based on the parking sensors, elevators' operation, HVAC System, vending machine operations, and many others. However, for this research and to have a limited scope, IBDMA proposed the autonomous monitoring and control of five different facilities including: HVAC system, luminosity levels, parking management, garbage management, and fire incident management. One example scenario could be if the luminosity sensor indicates that the luminosity levels in a particular location of the building are below a certain level, IBDMA proposes that this 'information' will help improve the luminosity levels of the room by turning the lights on in that location.

Generally, IBDMA proposes that the 'information' element also includes the visualization of the IoT data done in Power BI and Kibana, the analysis results generated by Apache Spark and the results for autonomous control of facilities within the smart building (prescriptive analytics), as shown in Fig. 10. The Spark program analyzes the data received from the IoT sensors, and based on the received data, it decides what action to take. For instance, the smoke detection sensor, sends a value indicating there is a fire in a certain location of the smart building, the Spark program will trigger the fire alarm and the fire extinguisher in that location. More details and specific use cases are provided in the next sections. As mentioned earlier, the "processes" integrate all the elements of the framework, hence the "processes" integrating "information" element to the rest of the elements of IBDMA include 'data visualization, 'data analysis and 'decision making' as represented in Fig. 17.

Facility

'Facility' is the final element of the framework. It includes numerous facilities of the smart building that are aimed to enhance residents' comfort, safety, and security. The 'information' generated from the IoT data enables the autonomous control of these 'facilities', as shown in Fig. 6. The facilities may include but are not limited to elevator maintenance and management, HVAC system, and garbage management. For this research, we consider the following five facilities: HVAC system, smart parking, smart garbage management, smart lighting, and smart fire management, as presented in Fig. 11. The target facilities that need to be controlled in the smart building should be identified and considered before defining the policies and requirements of the smart building

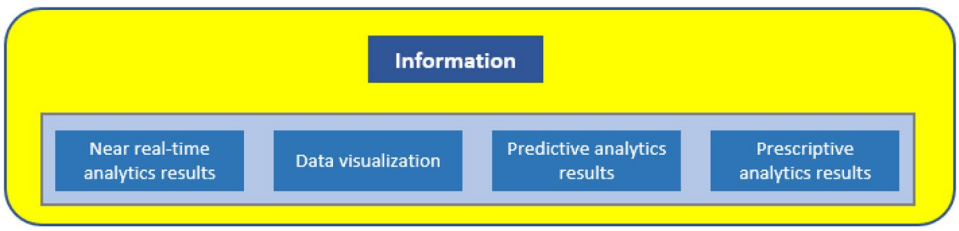

Fig. 10 The fourth element of IBDMA—Information

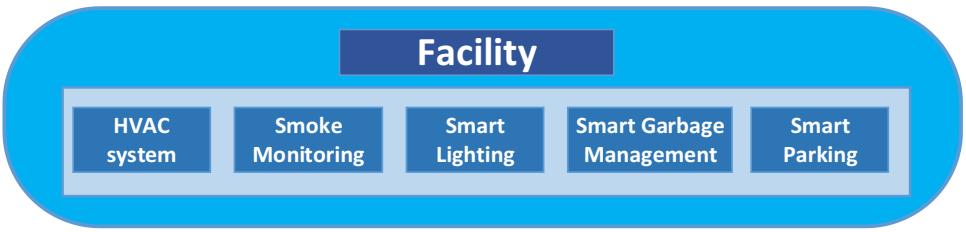

Fig. 11 The fifth element of IBDMA—facility

ecosystem. This reduces the possibility of encountering any major roadblocks in deploying the infrastructure for big data management of IoT-enabled smart buildings.

As discussed earlier, the 'process' element of IBDMA integrates all the elements of IBDMA, the process that integrates and control the facilities of the smart building based on the information generated by the system is called 'action'. The action process enables the autonomous control and management of the five facilities that fall under the scope of this research as represented in Fig. 17.

Reference Architecture Development Process

The development of the IBDMA reference architecture has been done in five iterations to ultimately reach to its current final state (iteration 5). In this section, we also provide the details of how the contextual-level elements of IBDMA are related to the physical level elements; this will be demonstrated in iteration 4 and 5 in the sub-sections below.

The five iterations for the development of the Big Data and Analytics architecture, for our use case, are discussed in detail below:

- Iteration 1 (Physical)
- Iteration 2 (Physical, real time)
- Iteration 3 (Virtual, smart building)
- Iteration 4 (Virtual, smart building, improved)
- Iteration 5 (Virtual, smart building, improved and finalized)

Iteration 1: In the first iteration, we extracted UTS (University of Technology Sydney) sensor data from UTS building 11 and imported that into RStudio by utilizing web-scrapping techniques. The data are available publicly on UTS's web portal. These data were graphed and plotted for various sensors and predictions were made about the sensor data using ARIMA (AutoRegressive Integrated Moving Average) [23] model. The implementation of graphs and ARIMA model was done in RStudio using R. This exercise was done to become familiar with the sensor types and the data available. It also helped us in familiarizing ourselves with tools and techniques that we could use for future research at that time. Figure 12 demonstrates the components and the steps that were taken in Iteration 1.

Iteration 2: In the second iteration, we prototyped a physical system consisting of an Arduino microprocessor board, physical sensors, and a linear actuator. The data were sent to HDFS (Hadoop Distributed File System) for storage from where it was imported to RStudio for predictive analytics. The problem with Iteration 1 was that the data available were only batched data. Our focus was on both batched as well as real-time data, so we decided to prototype a system with a couple of physical sensors connected to a microprocessor. The sensors considered for this iteration were temperature and smoke detection sensors. The sensors produced real-time data after regular intervals. A linear actuator was also connected to the system to simulate the behavior of a fire extinguisher scenario. The sensors generated the data in real time, the data were stored in HDFS, and from HDFS, we could perform predictive analytics as well as visualize it in Tableau [80, 81]. The data generated from the sensors were also analyzed in real time as it was generated. If the values generated by temperature sensor and the smoke detection sensor went above the threshold (simulating a fire scenario), the linear actuator got activated for 5 s, simulating that the fire extinguisher is activated to rectify the fire. The linear actuator would go off if the sensors read a value below the threshold. The steps and processes followed in Iteration 2 are shown in Fig. 13.

Iteration 3: In the third iteration, we focused on scaling up the architecture developed in the second iteration. For this iteration, we considered a smart building application scenario by introducing big data pipelining, storage and analysis tools. It was not possible to have access to a large number of physical IoT sensors and actuators in a lab environment, and thus, we decided to virtualize the IoT sensors by simulating

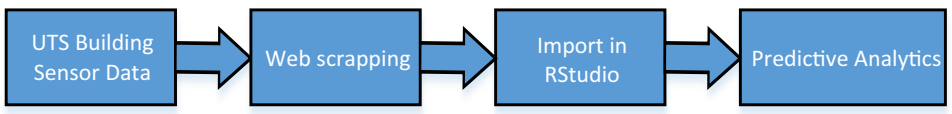

Fig. 12 Iteration 1—initial architecture design (Analysis of UTS Building 11 sensor data)

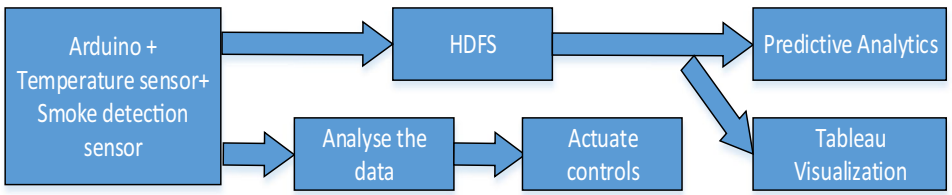

Fig. 13 Iteration 2—real-time data analysis and actuation using physical devices

the sensor data. Similarly, we simulated (virtualized) the actions taken based on the data received from the virtual sensors. This work has been published in [22]. The architecture developed for this iteration is shown in Fig. 14.

Data Sourcing

For this research, we virtualized the data generation from fifteen virtual sensors using a Python application. These 15 sensors include five (IoT) oxygen sensors, five smoke detection sensors, and five luminosity sensors deployed in a smart building. These 15 sensors are assumed to be deployed at five different locations (e.g., different rooms or floors) of the smart building in such a way that each location has a set of these three different sensor types, i.e., oxygen, smoke, and luminosity.

Data Ingestion and Storage

The data generated by these IoT sensors (source) are ingested into HDFS (sink) using an Apache Flume over a TCP (Transmission Control Protocol) port. For the implementation, we made use of the Cloudera [82, 83] Big Data platform (Virtual Machine for the Apache Hadoop environment) for extraction, ingesting, data pipelining, storing, and analyzing the data. For ingesting data into HDFS, Flume was the choice of tool because of Flume's robust integration with HDFS as compared to Kafka [84]. MQTT is a widely used protocol for IoT data; however, MQTT is primarily used as Machine-to-Machine protocol for transferring data between two physical systems. Since our goal is to move data to HDFS, we use Flume for data ingestion. There are a number of other tools available including Apache Beam, Apache Flink, Apache Storm, Apache NiFi, and Apache Ignite that can be used for streaming data analysis and event processing. However, for the purpose of this research and proof of concept prototype, we used Flume to ingest data and Apache Spark for its analysis.

Data Analysis and Building Control

For the analysis of data to enable decision-making, we developed an Apache Spark algorithm using PySpark [20]. The algorithm reads and analyzes the data from three different types of IoT sensors stored in HDFS in near-real time to enable effective decision-making. For instance, if the oxygen sensors generate data indicating a low oxygen concentration in a given location of the smart building, the Spark algorithm would in turn enable the HVAC System to turn ON to ensure that comfortable oxygen concentration levels are attained in that location. The system represents this by outputting "HVAC X turned ON" on the Cloudera terminal, where X represents the room or floor in a smart building. For the oxygen concentration threshold levels to turn the HVAC system ON or OFF, we defined the oxygen concentration threshold value of 14. On the other hand, if the oxygen concentration levels are above the threshold levels, the deployed infrastructure would represent this as "Oxygen level at X ok" on the Cloudera terminal, indicating that the oxygen concentration level in a particular location is above the comfortable threshold levels and that no further action is required to enable or disable the HVAC System. If the HVAC System was turned ON by the system due to low oxygen concentration levels and the oxygen concentration levels have become normal, the system would turn the HVAC system and will represent this by outputting "Fire alarm X turned ON" where X represents the room or level where smoke is detected.

Similarly, if during the data analytics process, a particular luminosity sensor detects lower than the minimum luminosity levels, the system will turn the lights ON that are located at that location. This is represented by the system by

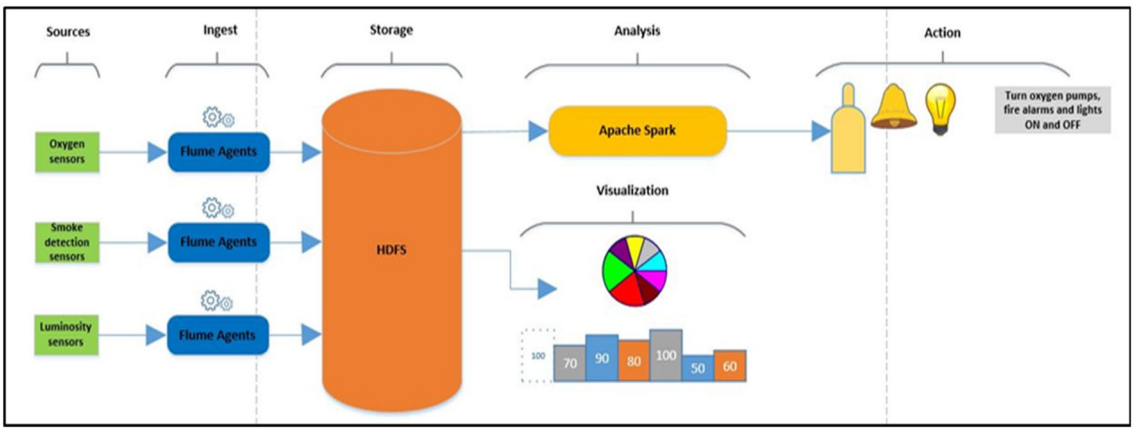

Fig. 14 Iteration 3—real-time data management, analysis and actuation for smart building

displaying "Lights at X turned ON" at the Cloudera terminal where X represents the particular room or level of the smart building.

Iteration 4: In iteration 4, we improved and extended the architecture to conceptualize the elements in terms of people, process, technology, information, and facility to link the contextual elements (Fig. 6) with the physical layer components (Fig. 14). The architecture for iteration 4 is shown in Fig. 15. The architecture developed in iteration 3 was scaled up and tested for a smart building application scenario by considering 1000 virtual IoT sensors.

Data Sourcing and Ingestion

For this iteration, we considered 200 of each of the five different types of IoT sensors which include oxygen sensors, smoke detection sensors, light sensors, parking spaces sensors, and garbage detection sensors.

Data Storage

Ten Flume agents were configured with IoT sensor data as the source and HDFS as the sink. The data were then visualized in Tableau.

Data Analysis and Building Control

Apache Spark was used to analyze the data in near-real time as it gets stored in HDFS. Based on the algorithm developed in PySpark, various messages were printed on the terminal screen simulating the feedback actuation behavior.

For oxygen sensors, if the value sent by a sensor is below a threshold, the PySpark algorithm prints out a message on the terminal stating the HVAC system associated with that particular oxygen sensor has been turned ON. For smoke detection sensors, if the value generated by a sensor exceeds a threshold (i.e., occurrence of a fire), the PySpark algorithm detects that and outputs a message on the terminal stating that the Fire Alarm connected at the location of that particular smoke detection sensor is turned ON. In case of luminosity sensors, if a particular luminosity sensor generates an output value below a threshold indicating it is dark, the PySpark algorithm outputs a message on the terminal stating that the lights associated with that particular luminosity sensor are turned ON. For the parking space sensors, if the value generated by a particular parking space sensor is a 1, the PySpark algorithm displays a message on the terminal stating that a car has been parked at that particular parking spot. For the garbage detection sensors, if the value generated by a particular sensor is above the threshold, the PySpark algorithm displays a message on the terminal stating that the garbage bin associated with the particular sensor which generated an above threshold value is full.

Iteration 5: The earlier iterations had limitations in terms of real-time data visualization and performing predictive analytics. In the fifth and final iteration, we worked on improving the architecture to enable real-time visualizations by introducing Elasticsearch [76, 85] and Kibana [77, 86]. We also introduced MS (Microsoft) Power BI [87, 87] for the visualization of data stored in HDFS. The main reason for introducing MS Power BI was because Power BI integrates well with R scripts. This integration provides the ability to do data analysis and predictive analytics within Power BI in an interactive way.

Moreover, we chose a hybrid model considering both batched as well as streaming data sources.

The high-level and generic reference architecture is presented in Fig. 16.

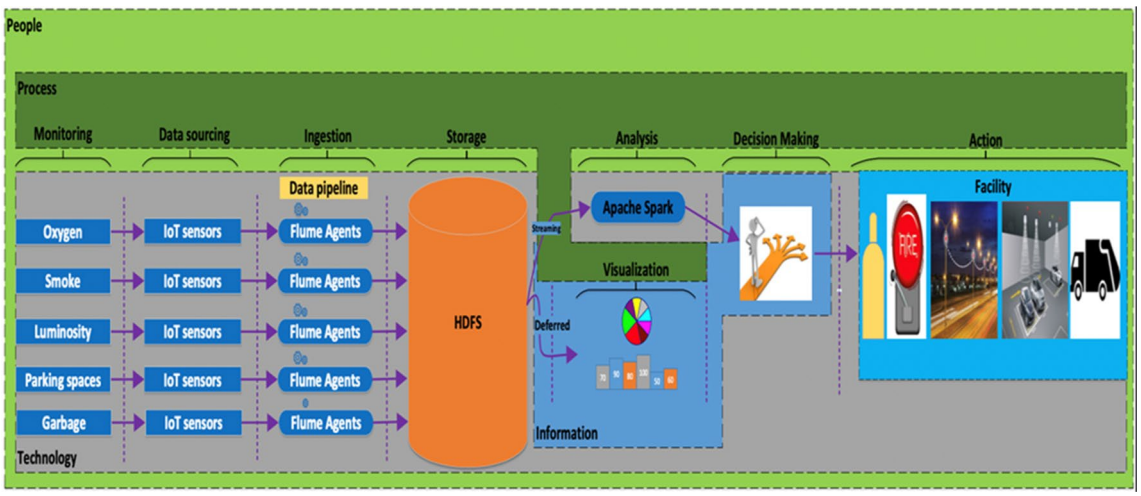

Fig. 15 Iteration 4—updated real-time data analysis and actuation architecture

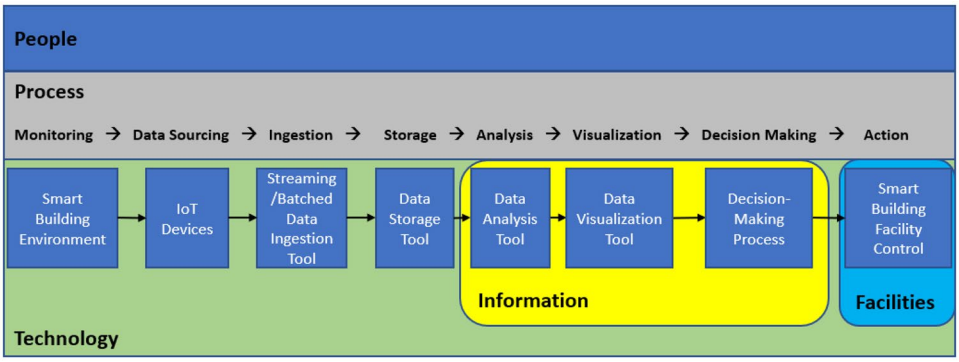

Fig. 16 Iteration 5—high-level architecture

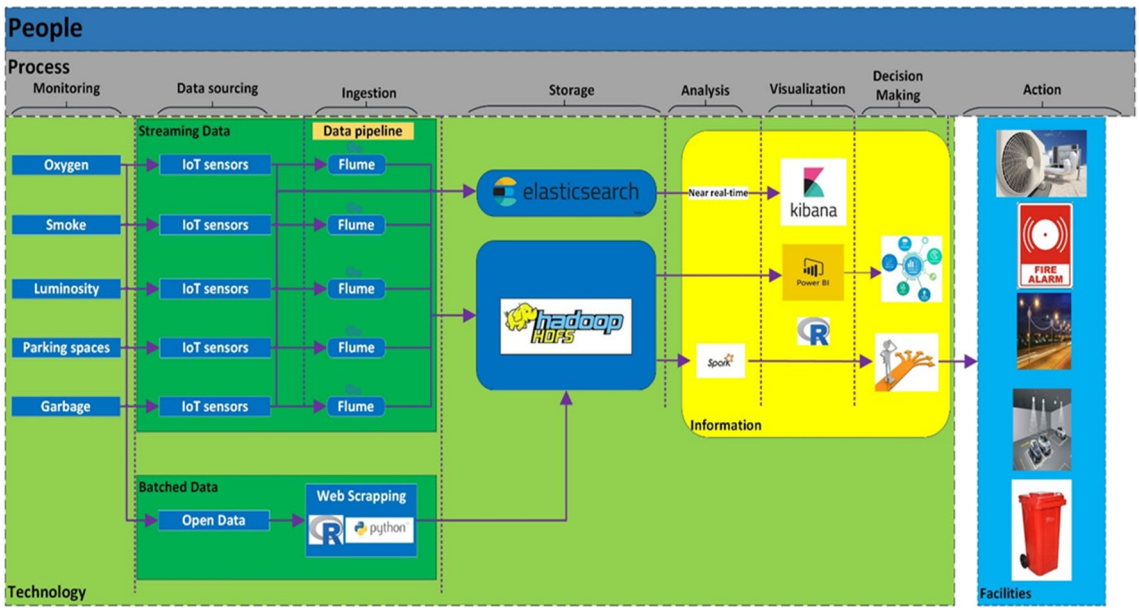

Fig. 17 Iteration 5—updated and improved near real-time data analysis and actuation architecture

However, for implementation and validation purpose, we chose specific tools, and the resultant architecture is presented in Fig. 17.

Batched Data

We chose open data as a batched data source for our architecture. These open data are scrapped using R (can also be done using Python) and ingested into HDFS.

Streaming Data

For streaming data sources, the virtual IoT sensors send the data to two sinks: (1) HDFS and (2) Elasticsearch. As the data from the IoT sensors land into HDFS, it is analyzed in near-real time by the Apache Spark algorithm to enable decision-making for the effective management and control of the five facilities earlier described within the smart building. These data once stored in HDFS are visualized in batches using Power BI.

Predictive and Near-Real Analytics:

For predictive analytics, we used R scripts to develop an ARIMA model within Power BI. For the second source, i.e., Elasticsearch, the data are indexed as it lands into Elasticsearch. Elasticsearch provides a data visualization plug-in called Kibana which enables the near-real-time visualization of IoT data.

The updated architecture is presented in Fig. 17. It shows how the five contextual elements shown in Fig. 6 are related to the physical elements of the framework. People are at the top-most level, which represent the stakeholders of the smart building environment such as building developers, building management, IT professionals, and residents of the building. Process element defines data-driven

Table 5 Elements in the IBDMA architecture and their purpose

Sr. No	Element	Process	Purpose
1	Flume	Ingestion	For ingesting streaming IoT data in Elasticsearch and HDFS via TCP/IP
2	Elasticsearch	Storage	Indexing streaming data to be visualized in Kibana
3	Kibana	Visualization	Visualizing streaming IoT data in near real-time
4	HDFS	Storage	Storing both batched and streaming data
5	Spark	Analysis/decision-making	Analyzing IoT data in near real-time to enable decision-making and actuation of smart facility
6	Power BI	Visualization	Visualizing batched and forecasted data
7	R	Ingestion/decision-making	Web scrapping and predictive analytics

processes, which are relevant to the smart building. This includes monitoring via sensors, data sourcing, ingesting, storing, analysis, visualization, decision-making, and finally actuation. The Technology element includes the technology stack including Flume, R, Elasticsearch, HDFS, Kibana, Spark, and Power BI. The information includes the near-real data visualizations in Kibana, Power BI dashboards for the IoT data visualization and the output of the decision-making process using Spark. Finally, the Facility element represents the facilities in the smart building, including HVAC systems, fire alarms, lights, parking spaces, and garbage spaces.

Table 5 summarizes the details of various big data tools used in the development of the IBDMA. It lists the processes in which each of these tools are used and the purpose of each of these tools in the IBDMA reference architecture implementation.

Reference Architecture Implementation

The proposed design of the IBDMA architecture is implemented for a smart building application scenario following the architecture as presented in Fig. 17. As shown in Fig. 17, we considered both streaming as well as static data. For streaming data, we created 1000 virtual IoT sensors. This is presented in the section marked Streaming Data in Fig. 17. These virtual sensors are implemented using a software application developed in Python. This application generates data from each sensor at regular intervals. Each sensor generated the sensor id and the value it measures from the building environment. The sensor id represents the location of the sensor in the building. The IoT application has a defined range of values for each type of sensor, and hence, the values are generated randomly between the ranges of values by the IoT application. This is done to keep the validation scenario simpler and to ensure that all possible scenarios are considered during the implementation and validation phases.

For the IoT sensors, we consider five different types of sensors. Out of the 1000 sensors, we simulate 200 oxygen sensors, 200 smoke detectors, 200 luminosity sensors, 200 parking spaces sensors, and 200 garbage detection sensors. It is assumed that these 1000 sensors are deployed at 200 distinct locations (including rooms or levels) of the smart building. We implement the example scenario using Cloudera VM (Virtual Machine) Hadoop distribution and used Python to create virtual sensor application to generate IoT data. The VM provides most the big data tools (Apache Flume, Apache Spark, HDFS, and Hive) required for the implementation of the IBDMA architecture. The other software packages (Pycharm IDE, Elasticsearch, and Kibana) were installed on the VM. The virtual sensor application forwarded the IoT data to two destinations. First destination is Elasticsearch where the data are indexed and stored, so Kibana can be used to visualize it. The second destination is multiple Flume agents ingesting data into HDFS. We configure ten Flume agents with each Flume agent serving 100 sensors. On ingesting the data into HDFS, it is analyzed in near-real time using PySpark (Python Apache Spark API) [20].

For the static data, we consider UTS smart building open data available publicly and ingested that in HDFS using R. This is presented in the Static Data section of Fig. 17.

The "process" element IBDMA framework integrates all other elements. The implementation of IBDMA architecture relies actually on the implementation of the processes, as shown in Fig. 6 and Fig. 17. Hence, we further the processes as shown in Fig. 8 to explain the implementation of the IBDMA architecture.

Monitoring

As shown in Fig. 17, 'Monitoring' is the first process in the IoT-enabled smart building. It includes the monitoring of various parameters of the IoT-enabled smart building ecosystem. For static data, we choose oxygen and gas detection sensors which monitor the oxygen levels and gas levels, respectively, within UTS Building 11.

For streaming data, the monitoring process is accomplished by the virtual IoT sensors developed using a python application which simulate the monitoring of oxygen levels, temperature levels (fire detection), luminosity levels, garbage levels, and parking spaces in the smart building.

Data Sourcing

Sourcing is the second process in the IBDMA implementation as presented in Fig. 17. As discussed above, we consider both static as well as near real-time streaming data as our data sources. For static data source, open data from UTS (University of Technology Sydney) smart building sensors were extracted from web using R and were stored in HDFS. These data comprised of historical data for two types of sensors for one of the levels/floors of UTS building 11. These sensor types include oxygen sensors and gas detection sensors.

As a streaming data source, real-time streaming data for 1000 virtual IoT smart building sensors were generated using Python virtual sensor software application. These data were ingested and stored in HDFS and we deployed big data tools to achieve the data ingestion task as explained in detail in the next section. Each virtual sensor had a unique sensor id by which it is identified. We identified the first 200 virtual sensors with sensor ids between 1 and 200 in our research as oxygen sensors, sensors with sensor ids between 201 and 400 (both inclusive) as smoke detection sensors, the sensors with sensor ids between 401 and 600 (both inclusive) as parking spaces sensors, the sensors with sensor ids between 601 and 800 (both inclusive) as luminosity sensors, and the sensors with sensor ids between 801 and 1000 (both inclusive) as garbage detection sensors. The Python virtual sensor application for data generation is developed using Pycharm IDE community edition [88, 89]. Figure 18 shows the logic flow diagram of the data generated from IoT sensors.

Figure 26 (Appendix) shows the screenshot of data generation part of the Python virtual sensor application.

As seen from Fig. 26 (Appendix), first, the required modules are imported including the socket and Elasticsearch modules. The TCP IP and the port are defined. The sensor class is defined, and the objects of the class including sensor id, sensor value, and the sensor location (room or floor of the smart building) are initialized. Then, in a while loop, the sensor id and sensor location are incremented by 1. The sensor value is generated randomly between a specified range of values for each different type of sensor. For example, Fig. 26 (Appendix) is the screenshot for the oxygen sensors, in which random values between 8 and 21 are generated which denotes the percentage oxygen concentration in air.

For this research, we have set up the data generation, such that at one time, 10 sensors will simultaneously be generating the data which are served by ten Flume agents running in parallel. The data are generated with 1 s interval, so the time interval between two consecutive data readings from a single sensor is 100 s which is found out to be a reasonable latency to report the smart building environmental conditions. This time interval can be reduced by slightly modifying the Python application.

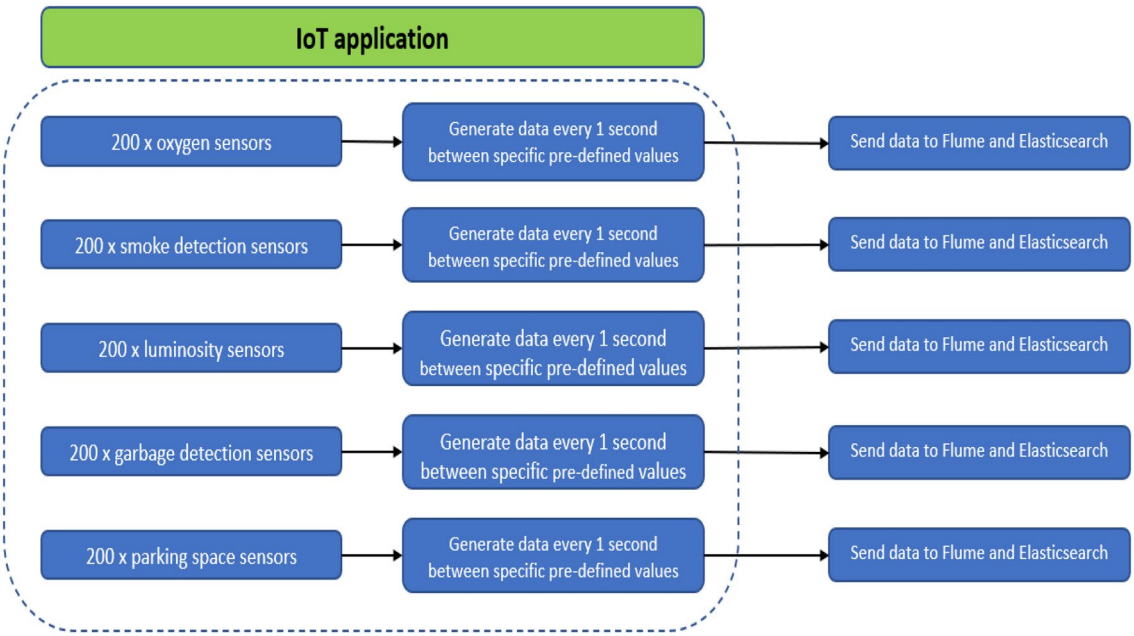

Fig. 18 Data sourcing—logic flow diagram

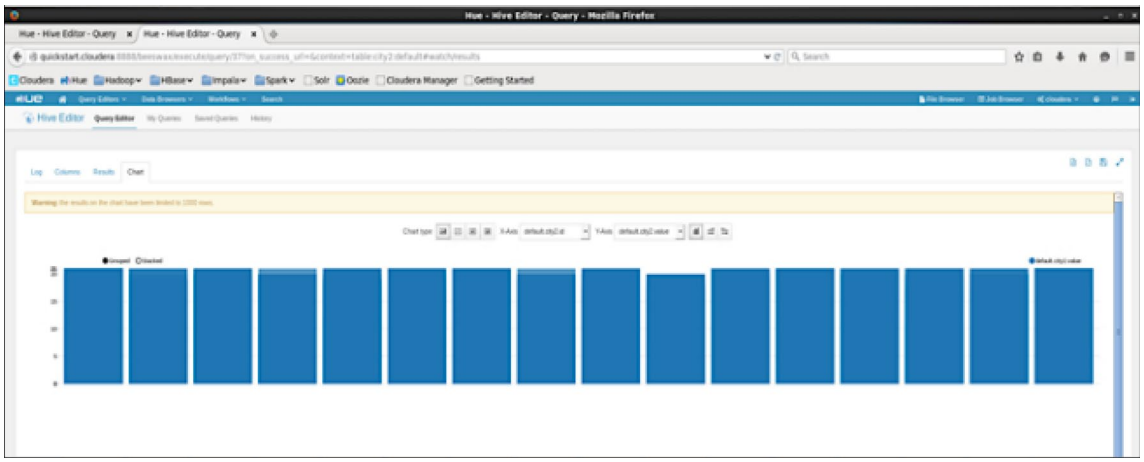

Fig. 19 Data visualization in Cloudera using Hive tables

The virtual sensor application pushes data to two destinations: (1) to Elasticsearch to enable near real-time data visualization using Kibana (2) to Flume agents to enable near-real-time ingestion of data into HDFS. To store and index data in Elasticsearch, it provides Python API and we used this API to store and index data into Elasticsearch. In the virtual sensor application, an Elasticsearch document is defined which includes sensor id, sensor value, sensor location and the time of generation of data. This document is then sent to Elasticsearch index named "iot" as can be seen in Fig.26 (Appendix).

Figure 26 (Appendix) shows the data generation code for first 100 oxygen sensors. For the case of oxygen sensors, the values are randomly between 8 and 21, where these values denote the percentage concentration of oxygen in air. If the values are above 14, oxygen levels are considered normal. For smoke detection sensors, same range of values are generated randomly with values above 14 denoting a possible fire scenario. For the case of parking space sensors, each sensor outputs a high (1) or a low (0) value denoting a particular parking space is full or empty. For luminosity sensors, random values between the range of 8 and 21 are generated with values above 14 are considered normal and values below 14 representing luminosity levels below normal. For the case of garbage detection sensors, the sensors generate a high (1) or low (0) to represent if a garbage bin is full or empty.

In the last part of the code, the sensor id is reset once the sensor id reaches from 1 to 100. Therefore, the value for sensor 1 is generated and the cycle repeats. Finally, the TCP connection and the connection to Elasticsearch cluster are closed when the program is exited.

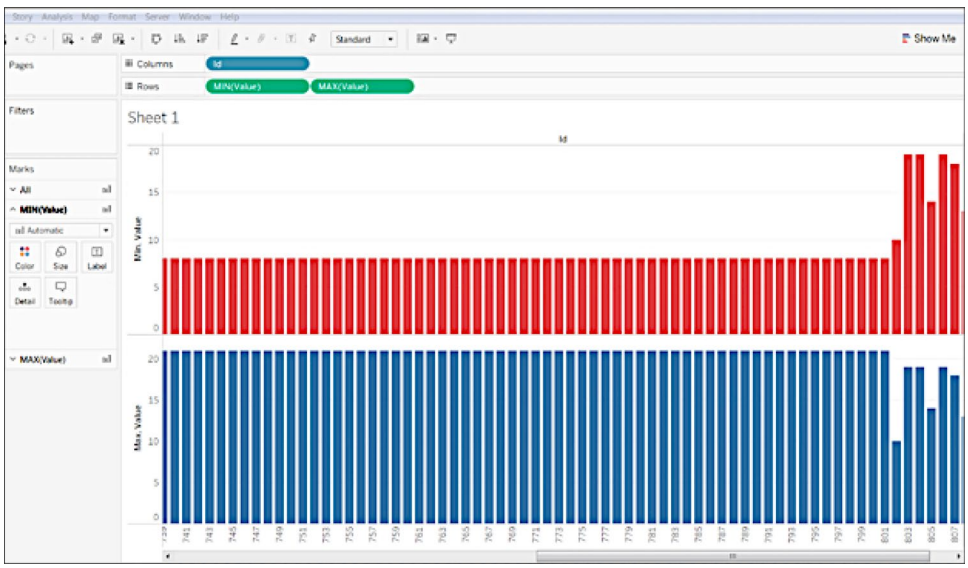

Fig. 20 Data visualization in tableau (min and max values displayed top and bottom)

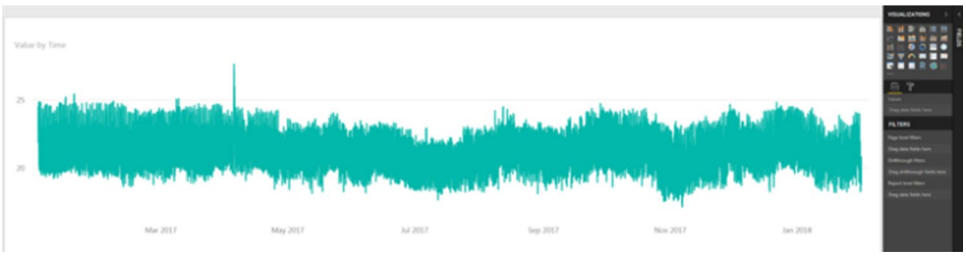

Fig. 21 Temperature sensor data visualization in MS Power BI

Data Ingestion

Data ingestion is the third process, as shown in Fig. 17. For static data, the UTS building data once extracted in the .csv format are ingested and stored in HDFS.

For streaming data, we deploy Apache Flume to ingest IoT sensor data into HDFS. Ten Apache Flume agents are configured and used for data ingestion. These agents are configured to listen to ten different TCP ports as specified in the Virtual sensor application to reduce time latency, increase throughput of the data and to prevent loss of data. The configuration of these Apache Flume agents is done, such that the virtual IoT data generated from the Python virtual sensor application acts as the source and the HDFS acts as the sink to store the data into HDFS as soon as it arrives from the virtual sensors.

Figure 27 (Appendix) shows the contents of one of the Flume configurations files out of ten configuration files. The configuration file has three key elements: Source, Sink, and Channel. The 'roll-over interval' for the Flume agent was unchanged to the default 30 s interval setting which means that each Flume agent will roll over files after every 30 s, finish writing to it and create a new file in HDFS every 30 s as. tmp file. This .tmp file gets converted to a permanent file after the 30 s have elapsed. Specifying 0 for roll-over interval will disable rolling and will case all events to be written to a single file.

As seen from Fig. 27 (Appendix), the Flume agent name defined in the configuration file is 'a1'. There are three key components in a flume configuration file, i.e., source, sink, and channel. 'Source' binds to the incoming source of data, while 'Sink' binds to the destination where the data need to be stored. 'Channel' as the name suggests provides a channel to transfer data from source to the sink. The 'Source' defined in the configuration file is a netcat source which binds to TCP IP 127.0.0.1 on port 5005. The sink is an HDFS sink with a path hdfs://quickstart.cloudera:8020/user/cloudera/virtualsensor1. The channel is a memory channel with a capacity and transaction capacity of 1000 and 100, respectively. Capacity defines the maximum number of events stored in the channel. Transaction capacity defines the maximum number of events the channel will take from a source or give to a sink per transaction.

For batched data, data ingestion process involves web-scrapping the open data using R or Python. We used R to download the data to our local disk and manually uploaded the extracted data into HDFS.

Data Storage (Big Data Management)

Data storage is the fourth process, as shown in Fig. 17. For static data, we downloaded the IoT sensor data from UTS building 11 by performing 'web-scrapping' using R. The downloaded data were first to our local disk as one single

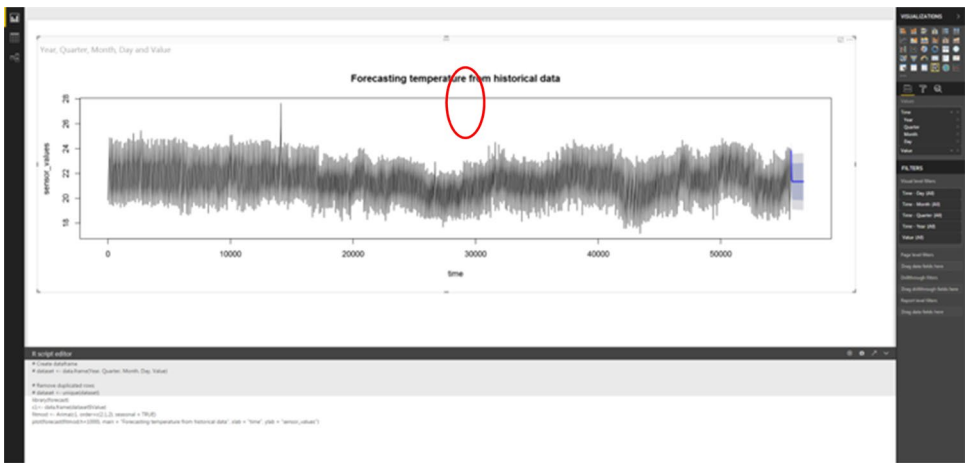

Fig. 22 Forecasting sensor data in power BI using R

Fig. 23 Near real-time sensor data visualization in Kibana

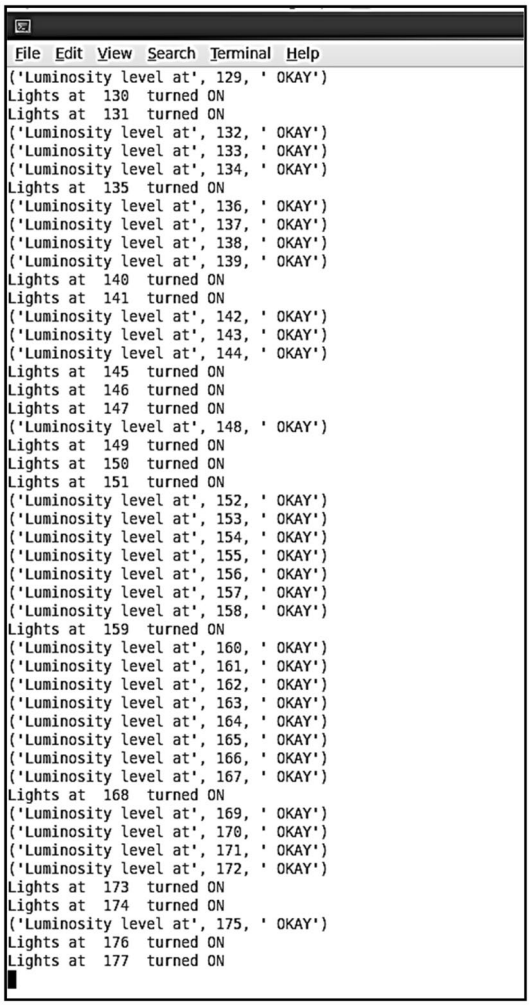

Fig. 24 Smart building control messages

text file and then manually uploaded to HDFS at a specified folder after performing some data manipulation on the downloaded data. The static data included data from temperature, luminosity, humidity, and oxygen sensors. The data comprised of sensor ids, the value generated by the sensors, and the timestamp separated by commas.

For the streaming data, we store data both in Elasticsearch as well as HDFS. The reason for storing data into Elasticsearch is to enable real-time visualization using Kibana. HDFS is used as the other storage destination to enable near real-time analysis using Apache Spark.

Elasticsearch is a scalable distributed search engine which provides powerful APIs to enable extremely fast data search for data discovery applications on an enterprise grade. Data are stored in Elasticsearch in the form of indexed documents using APIs where Elasticsearch adds a searchable reference to the document in the cluster's index. The documents can then be retrieved using the Elasticsearch API. To manage Elasticsearch and the data stored in Elasticsearch, we used Kopf plug-in based on JavaScript + AngularJS + jQuery + Twitter bootstrap which provides an easy-to-use web-based administration tool to manage Elasticsearch cluster.

The IoT sensor application sends the data to Elasticsearch where it is indexed and stored in the form of documents. Kibana is then used to visualize the data in near-real time. The IoT sensor application also sends data to HDFS via ten Flume agents as discussed in previous section. Flume has a default naming convention for storing files into the HDFS which includes the timestamp. Since the roll-over time for all the ten Apache Flume agents was 30 s, as discussed in the previous section, after every 30 s, a new .tmp file was

created in the specified folder within HDFS for each Flume agent, and after those next 30 s have elapsed, this .tmp file gets converted automatically to a permanent file on HDFS.

The data generated from the Python IoT sensor application contain sensor id, sensor value, and the location of the sensor. Both the storage destinations, i.e., Elasticsearch and HDFS, store the data generated by the Python IoT sensor application.

Data Analysis (Big Data Analytics)

Data analysis is the fifth process in the IBDMA architecture as presented in Fig. 17. To analyze the IoT sensor data in near real time, we used PySpark—Spark Python API (Application Programming Interface) for this research. For the streaming data, we develop an algorithm within PySpark which monitors the incoming data in near-real time as it is ingested into HDFS. Depending upon data values generated by the virtual sensors, the algorithm detects whether there is a need to activate any controls within the smart building or not. If, for instance, a particular smoke detection sensor sends out a value that is too high indicating a possible fire scenario, the PySpark algorithm detects it and outputs a text message that the fire alarm located in the vicinity of the sensors has been activated. In case of batched data, the PySpark algorithm reads the whole file and outputs a detailed message on the terminal for each sensor indicating whether there was a problem detected for any of the sensors or whether all the values received were within range. This helps to identify the problems within the system and to improve the comfortability and safety of the residents. The PySpark code for data analysis is provided in the github repository [90].

Data Visualization

Data visualization is the sixth process in the IBDMA architecture as presented in Fig. 17. The data analysis process is followed by data visualization process. For real-time streaming data, we considered both near real time as well as batched visualizations. For near real-time visualization, we use Kibana which integrates with Elasticsearch and produces near real-time visualizations of the received data. These visualizations help in identifying the problems in the smart building in near real time without going through chunky data sets or running algorithms on the data. For batched visualization, we concatenated multiple files stored in HDFS generated as a result of the data storage process into a single file to make the visualization process easier. We adopted various tools to do the batched visualizations. First, we chose the Cloudera built-in visualization tool by importing the data in Hive table. Hue was used to create Hive tables was within Cloudera and the data stored in HDFS were imported into the table. The visualization done in Hue is shown in Fig. 19. The limitation of doing visualization in Hive within Cloudera is that we cannot visualize more than 1000 rows of data.

Second tool we adopted for data visualization is Tableau. Tableau has a connector which allows it to get connected to the HDFS data. Figure 20 shows snapshots of the data visualization done in Tableau. As compared to Hive data visualization, the visualization in Tableau is much more flexible as you can customize the results in a better way to create more useful visualizations. Figure 20 shows the minimum and maximum values generated by a particular sensor in a top–bottom view with minimum values on the top in red and maximum values on the bottom in blue.

The third tool we adopted for batched data visualization was MS Power BI. We found MS Power BI to be the best tool for our research as it has an easy-to-use interface, it integrates well with HDFS, and we could write R programming scripts within MS Power BI. The advantage of using R scripts within Power BI was that we could develop and run predictive models within Power BI. The visualization done in Power BI for open batched data for a temperature sensor for the last 13 months is presented in Fig. 21.

The data from the sensor were generated after every 7 min. We developed an R script to forecast the next 1000 values of the temperature sensor which corresponds to approximately the next 5 days. We chose an ARIMA model to do the predication. The forecast results are presented in Fig. 22.

The discussion on the selection of the appropriate model, its fine tuning, and evaluation are beyond the scope of this research. However, the code we used to generate a simple Arima model and to forecast the next 1000 values for the temperature sensor is presented in Fig. 28 (Appendix). Because temperature sensor values depend on the seasonal factors, i.e., quarter of the year, month, or days of a week, we have set seasonality is equal to 'true' in our ARIMA model. The modeling can be extended to identify and predict various parameters of the smart building, but we keep this discussion beyond the scope of this paper.

Power BI is a great visualization tool when it comes to visualizing extracted or static data. However, to visualize near real-time IoT streaming data, Power BI and lot of tools available in the market fall short. That is why, we chose Elasticsearch and Kibana to visualize streaming data in near-real time.

Once data are indexed in Elasticsearch as explained in Sect. 6.4, Kibana is an open-source data visualization tool and is used to develop near real-time visualizations and dashboards for the documents indexed into Elasticsearch cluster. Figure 23 shows the near real-time data visualization in Kibana for one particular IoT sensor; it shows the minimum and maximum values received from that particular IoT sensor.

Decision-Making

Decision-Making is the seventh process in the IBDMA architecture, as shown in Fig. 17. The 'decision making' process includes understanding and evaluating results from the 'Data Analysis' and 'data visualization' processes. It involves analyzing and visualizing predictive analytics results obtained by integrating R predictive modeling scripts with the visualization of the virtual IoT sensor data done in MS PowerBI. The decision-making process also involves analyzing the results of the PySpark algorithm. The decision-making process helps the smart building stake holders in making better data-driven decisions to improve the comfortability, safety, security, efficiency, and safety of the smart building.

Action: Smart Building Control

Action is the last process in the IBDMA architecture, as shown in Fig. 17. Once a decision is made based on the data analysis and data visualization results, the smart building facilities are controlled autonomously by the IBDMA. This is accomplished by triggering the appropriate controls of the facilities deployed in the smart building as a result of the data produced by the corresponding IoT sensor. To simulate this behavior, the system displays the results in the form of textual output displayed on the Cloudera terminal.

The oxygen concentration is controlled and maintained by analyzing the data generated from the Oxygen sensors deployed in the smart building. As soon as the data from the oxygen sensor are ingested into HDFS, the Apache Spark algorithm analyzes these incoming data, and if the value received by a sensor is below 14 (which reflects the oxygen concentration level), the Spark algorithm considers it a value below the appropriate threshold and triggers the corresponding HVAC ON. For the purpose of our research, we simulate this by displaying "HVAC system X turned ON" on the Cloudera terminal, where X represents a particular location in the building. Now, when the HVAC system stays ON for some time, the Oxygen concentration levels in that particular location will start going up and soon, the levels will be in the acceptable range. When this happens, the system turns the HVAC system OFF. For the purpose of this research, we simulate this behavior by displaying "HVAC system X turned OFF". On the other hand, if another oxygen sensor deployed at another location within the smart building already is reading acceptable oxygen concentration levels, the proposed IBDMA reference architecture does not take any triggering actions. In this case, the PySpark algorithm outputs "Oxygen level at X OKAY", depicting that oxygen levels detected by sensor id X are okay.

To maintain a safe and productive environment for the residents, we need to ensure that the smoke detection sensors are

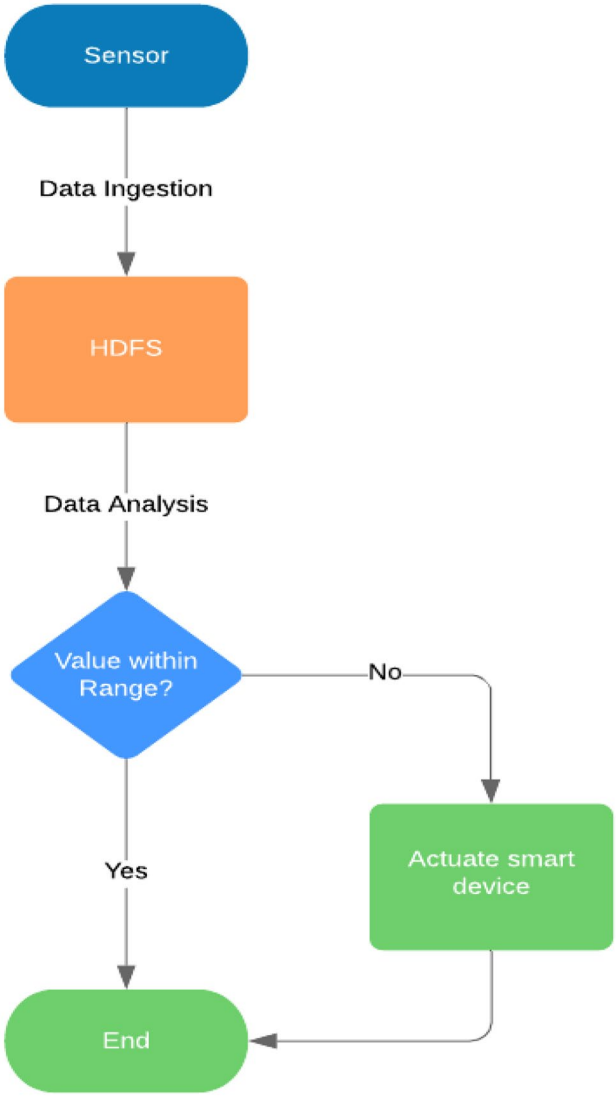

Fig. 25 Flowchart of actuation process

installed to monitor any smoke or fire hazards. The smoke detection IoT sensors continuously monitor the environment for a fire hazard and in case of a fire send a value above a predefined threshold value indicating a fire in the smart building. When this happens, the proposed architecture triggers the corresponding fire alarm location in the same location as the sensor where fire was detected. For the purpose of our research, the Spark algorithm outputs "Fire alarm X turned ON" on the Cloudera terminal simulating the behavior of turning a fire alarm ON where X represents the location or room of the smart building where the sensor detected smoke or fire. On the other hand, in normal situations where there is no fire or smoke detected, the IBDMA architecture will not produce alerts and will keep performing normally by outputting "No fire at X" on the Cloudera terminal. The system will do this for all the locations of the smart building.

To monitor and effectively maintain the parking spaces of the smart building, parking sensors are deployed in the smart building. When a parking space is filled, the IBDMA architecture displayed "Parking X is occupied" where X represents the parking spot which has been occupied by a vehicle. When the vehicle leaves the parking spot, the architecture prints out the text saying, "Parking X is empty". These data can be collected and analyzed to make a decision on how to improve the parking areas for the residents.

The luminosity sensors deployed in the smart building monitor the luminosity levels in numerous locations of the smart building. The data generated by these sensors once ingested into HDFS are analyzed by the Spark algorithm, and if the value sent by the sensor is below a threshold luminosity level, the proposed architecture will turn the lights in that location ON. For the purpose of our research, we simulate this by outputting "Lights at X turned ON" on the Cloudera terminal where X represents the sensor id or location of the smart building where low luminosity levels were detected. On the other hand, if the luminosity levels in a particular room are already within the acceptable range, the proposed architecture displays "Luminosity level at X OKAY", illustrating that luminosity levels at that particular location are okay, and hence, no action is required. However, in cases where the lights need to be turned OFF at a particular location, the proposed architecture is capable of handling this and simulates this be displaying "Lights at X turned OFF" on the Cloudera terminal.

To effectively and efficiently manage the garbage in the building, garbage detection sensors are deployed in the smart building which monitor the garbage bin levels of the bins placed in the garbage areas of the smart building. When a garbage bin becomes full, the proposed architecture generates a warning by displaying, "Garbage at X is Full" where X is the location of the garbage detection sensor where it detected a garbage bin is full. For all the garbage bins which have space for more garbage, the proposed architecture continuously monitors the garbage levels and prints out "Garbage at X has space" for all the bins in the building. This enables more effective and efficient management of the garbage within the smart building. Further decisions about the installation of more garbage bins and the garbage collection times can also be made by looking at the trend of the data generated by garbage detection sensors.

Figure 24 demonstrates the outputs displayed by the reference architecture on the Cloudera terminal as a result of the analysis if incoming data are from the IoT sensors. The actuation process is presented in a simplified flow chart as presented in Fig. 25.

Framework Evaluation Results

IBDMA framework and reference architecture was evaluated by testing five test cases against the 13 ECs presented in Fig. 4 of "Research Method". Streaming data from virtual sensor application were generated and the Cloudera big data platform was set up as explained in the previous section. The details of the test cases are mentioned below in Tables 6, 7, 8, 9 and 10.

Table 6 Test case 01—detection of oxygen levels

Test case 01	Detection of low oxygen level and autonomously activating HVAC system
Context	Oxygen levels fall below the human comfort levels at a specified location in the building
Problem	Low oxygen levels in the smart building may get unnoticed by the smart building management that could result in the discomfort of the residents and could potentially prove fatal
Solution	Oxygen level falls down and is detected by the IBDMA reference architecture. The management is notified, and the HVAC is autonomously activated within two minutes
Test metrics	Test duration: 60 min, number of records in data: 60, detection measure: The terminal displayed the message saying the HVAC turned ON when the value of the sensor fell below the threshold
Description	Oxygen levels in the smart building fall, the oxygen sensor senses that the levels of oxygen at a particular location have fallen below the threshold levels and sends the data to HDFS via Flume. The system detects the low oxygen levels, activates the associated HVAC system installed at that particular location and notifies the Smart Building Management within two minutes. The system autonomously turns the HVAC system off when the oxygen levels are in the acceptable range
Consequences/improved performance metrics	The residents enjoy comfortable environment. The smart building management are notified within two minutes if the levels fall down below the threshold levels and the HVAC system is turned ON or OFF based on the oxygen level
'As-as' vs 'to-be' solution	Literature is scarce on the integrated topic of near real-time alerts and visualizations for building management. Moreover, the ability to trigger the HVAC system autonomously within required time (e.g., two minutes) has not been discussed. The proposed reference architecture addresses both the above-mentioned challenges

Table 7 Test case 02—detection of fire

Test case 02	Detection of fire and alerting local community
Context	A fire erupts at a specified location in the building
Problem	Fire erupts in the smart building and may get unnoticed by the smart building management for a longer period potentially resulting in the loss of infrastructure, investment and lives
Solution	Fire erupts and is detected by the IBDMA reference architecture. The management including the Fire Brigade is notified within two minutes of the Fire eruption and a Fire Alarm at that location is turned ON and the fire extinguisher located in that area is triggered
Test metrics	Test duration: 60 min, number of records in data: 60, detection measure: The terminal displayed the message saying the Fire Alarm turned ON when the value of the sensor went above the threshold
Description	Fire erupts in a smart building; the IoT smoke detector sensor senses fire and sends the data to HDFS via Flume. The system detects fire, activates the associated fire alarm installed at that particular location alerting the local community and notifies the smart building management including the relevant fire brigade team within two minutes. The fire brigade team then acts on the notification by eradicating fire at the location
Consequences/improved performance metrics	The local community is alerted by activating the fire alarm so that they can move to safe areas. The fire brigade and building management are notified within two minutes so that they can respond to the fire to minimize loss of infrastructure, investment and precious lives
'As-is' vs 'to-be' solution	Literature is scarce on the topic of near real-time alerts and visualizations for building management. Moreover, the ability to trigger the fire extinguisher autonomously (e.g., within two minutes) has not been discussed before. The proposed reference architecture addresses both the above-mentioned challenges

Table 8 Test case 03—detection of luminosity

Test case 03	Detection of low luminosity level and autonomously activating smart lights
Context	Luminosity levels fall below the human luminous comfort levels at a specified location in the smart building
Problem	Low luminous levels in the smart building may get unnoticed by the smart building management that could not only result in the discomfort of the residents but could also prove to be a safety hazard for the residents
Solution	Luminosity level falls and is detected by the IBDMA architecture. The management is notified, and the smart lights are autonomously activated within two minutes
Test metrics	Test duration: 60 min, number of records in data: 60, detection measure: The terminal displayed the message saying the Lights turned ON when the value of the luminosity sensor fell below the threshold
Description	The luminosity sensor detects that the luminosity levels at a specified location have fallen below the threshold levels and sends the data to HDFS via Flume. The system detects the low luminous levels, activates the associated smart lights installed at that location and notifies the smart building management within two minutes. The system autonomously turns the smart lights off when the luminosity levels are in the acceptable range
Consequences/improved performance metrics	The residents enjoy comfortable luminous levels. The smart building management are notified within two minutes if the levels fall below the threshold levels
'As-is' vs 'to-be' solution	Literature is scarce on near real-time alerts and visualizations available to the building management. Moreover, the ability to control lights autonomously has not been covered. The proposed reference architecture addresses both the above-mentioned challenges

Table 9 Test case 04—detection of parking space usage

Test case 04	Detect if parking lot becomes full and alert the residents
Context	A parking space gets filled with car in the smart building
Problem	Parking space in the smart building becomes full and may get unnoticed by the residents for a longer period resulting in a lot of inconvenience for the residents
Solution	Parking space becomes full and is detected by the IBDMA architecture. The smart building residents are notified within two minutes of the parking space becoming full and are notified to move to alternate parking areas
Test metrics	Test duration: 60 min, number of records in data: 60, detection measure: The terminal displayed the message saying the Parking space is occupied when the value of the sensor was '1'
Description	Parking lot becomes full in the smart building the IoT parking lot sensor senses it and sends the data to HDFS via Flume. The system alerts the local community and notifies the smart building management within two minutes
Consequences/improved performance metrics	The residents are alerted so that they can move to alternative parking spaces or areas. The building management is notified within two minutes so that they can respond to the growing needs of the residents by planning out more parking lots in the areas
'As-is' vs 'to-be' solution	Literature is scarce on near real-time alerts and visualizations available to the building management. Moreover, the ability to address parking lots autonomously is also something new discussed in this paper. The proposed reference architecture addresses both the above-mentioned challenges

Table 10 Test case 05—detection of garbage

Test case 05	Detect if garbage bin becomes full and alert the residents
Context	A garbage bin becomes full in the smart building
Problem	Garbage bin in the smart building becomes full and may get unnoticed by the smart building garbage management team for a longer period resulting in a lot of inconvenience for the residents
Solution	Garbage bin becomes full and is detected by the IBDMA reference architecture. The smart building management is notified within two minutes of the garbage bin becoming full and the residents are notified to wither wait or throw their garbage at alternative locations
Test metrics	Test duration: 60 min, number of records in data: 60, detection measure: The terminal displayed the message saying the garbage bin is full when the value of the sensor was '1'
Description	Garbage bin becomes full in the smart building; the IoT garbage detection sensor senses it and sends the data to HDFS via Flume. The system alerts the local community and notifies the smart building management within two minutes. The smart building garbage management team sends the garbage collector to empty the garbage at the location
Consequences/improved performance metrics	The residents are alerted so that they can throw the garbage to alternative garbage bins. The smart building garbage management team is notified within two minutes so that they can respond to the growing needs of the residents by sending garbage collectors more often to the specified location
'As-is' vs 'to-be' solution	Literature is scarce on near real-time alerts and visualizations available to the building management. Moreover, the ability to manage garbage autonomously is addressed in this paper as well. The proposed reference architecture addresses both the above-mentioned challenges

The five test cases discussed above were evaluated against the 13 ECs presented in Fig. 4 and the results of the evaluation are presented in Table 11.

Conclusions and Future Work

This paper presented the IBDMA framework and its reference architecture, which was developed and evaluated for a smart building example scenario. The framework has five key integrated components: (1) People, (2) Process, (3) Technology, (4) Information, and (5) Facility. The reference architecture was evaluated using five iterations for the smart building scenario. The final iteration was implemented with a total of 1000 sensor involving 200 virtual oxygen sensors, 200 virtual smoke/hazardous gas detectors, 200 luminosity sensors, 200 parking sensors, and 200 garbage detection sensors. The evaluation was performed following the DSR approach [65] using the five different use cases against thirteen evaluation criteria taken from the literature. The results of the evaluation indicate how a systematic framework like IBDMA can assist for managing and analyzing the near real-time big data streams for smart buildings. It also demonstrates that and how various elements of a smart building including IoT sensors and control systems can be autonomously monitored and controlled in near-real time using the proposed IBDMA framework. Thus, the proposed

Table 11 Evaluation results for evaluating each test case against the 13 ECs

Test case	Evaluation criteria	Pass ✓/fail✗
01	EC1	✓
	EC2	✓
	EC3	✓
	EC4	✓
	EC5	✓
	EC6	✓
	EC7	✓
	EC8	✓
	EC9	✓
	EC10	✓
	EC11	✓
	EC12	✓
	EC13	✓
02	EC1	✓
	EC2	✓
	EC3	✓
	EC4	✓
	EC5	✓
	EC6	✓
	EC7	✓
	EC8	✓
	EC9	✓
	EC10	✓
	EC11	✓
	EC12	✓
	EC13	✓
03	EC1	✓
	EC2	✓
	EC3	✓
	EC4	✓
	EC5	✓
	EC6	✓
	EC7	✓
	EC8	✓
	EC9	✓
	EC10	✓
	EC11	✓
	EC12	✓
	EC13	✓
04	EC1	✓
	EC2	✓
	EC3	✓
	EC4	✓
	EC5	✓
	EC6	✓
	EC7	✓
	EC8	✓
	EC9	✓
	EC10	✓
	EC11	✓
	EC12	✓
	EC13	✓
05	EC1	✓
	EC2	✓
	EC3	✓
	EC4	✓
	EC5	✓
	EC6	✓
	EC7	✓
	EC8	✓
	EC9	✓
	EC10	✓
	EC11	✓
	EC12	✓
	EC13	✓

IBDMA framework is unique in a way that it integrates together end-to-end sensing, data management, autonomous actuation capabilities with the ability to do near real-time data management and analytics, which have not been discussed before, in particular in the smart building context. This marks the important contribution of this research. The framework is applicable to find a number of applications and can be further extended to other smart environments, such as smart homes, smart cities, and smart grids. As part of this research, we considered all the sensors data values or parameters independently. This research can be extended by considering the inter-dependence of these sensors or parameters on each other to extract further insights from the data.

IBDMA Framework Evaluation Repository

The configuration files and the code for the IBDMA reference architecture are available at the GitHub repository [90].

Appendix

See Figs. 26, 27, 28.

```
import socket
import time
import random
import datetime

from elasticsearch import Elasticsearch     # import elasticsearch module
es = Elasticsearch()

TCP_IP = '127.0.0.1'        # Define the TCP IP
TCP_PORT = 5005             # Define the TCP Port
BUFFER_SIZE = 500

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)          # Create an INET streaming socket
s.connect((TCP_IP, TCP_PORT))                                  # Connect to the server on TCP_PORT

# define the Sensor class
class Oxygen:
    id = 1                                  # Initialize sensor id
    value = 1.00                            # Initialize sensor value
    def description(self):
        desc_str = "The value of sensor %d is %.2f" % (self.id, self.value)
        return desc_str

# Initialize sensor class
sensor = Oxygen()

# Define and initialize variables
sensor.value = 0
sensor.id = 1

# Start generating random values for the sensor class
while(1):
# sensors 1-200 are Oxygen sensors
# sensors 200- 400 are Smoke detection sensors
# sensors 400-600 are Parking space sensors
# sensors 600- 800 are Luminosity sensors
# sensors 800-1000 are Garbage sensors
      sensor.value = (random.randint(8,21))         # Generate the values for Oxygen in the range from 8 to 21
      t = str(sensor.id).encode()                   # Convert the id to string so it can be sent to the TCP connection
      print (sensor.id)                             # Print the sensor id
      s.send(t)                                     # Send the id on the TCP connection
      s.send(",")                                   # Send a comma to separate id and value
      m = str(sensor.value).encode()                # Convert sensor value to string so it can be sent to TCP connection
      s.send(m)                                     # Send the id on the TCP connection
      s.send(",")                                   # Send a comma to separate value and postcode
      s.send("\n")                                  # Send a newline character so the next set of values be printed on next line
      sensor.id = sensor.id+1                       # Increment the sensor id by 1
      time.sleep(10)                                 # Wait for 1 second to generate next value
      # Create a document to be sent to elasticsearch
      doc = {
          'sensorid': sensor.id,
          'value': sensor.value,
          'timestamp': datetime.datetime.now(),
      }
      res = es.index(index="iot", doc_type='smart_building', body=doc)   # Index the document in elasticsearch
      print(res['created'])                                              # Print if indexed successfully
      if(sensor.id==101):                           # Reset the sensor id and postcode when it reaches 100 sensors
            sensor.id = 1                           # Reset sensor id
s.close()
```

Fig. 26 Sensor data generation code

Fig. 27 Flume configuration file

```
cloudera@q
File Edit View Search Terminal Help
# sensor100.conf: A single-node Flume configuration

# Name the components on this agent
 a1.sources = r1
 a1.sinks = k1
 a1.channels = c1

# Describe/configure the source
 a1.sources.r1.type = netcat
 a1.sources.r1.bind = 127.0.0.1
 a1.sources.r1.port = 5885

# Describe the sink
 a1.sinks.k1.type = hdfs

# Use a channel which buffers events in memory
 a1.channels.c1.type = memory
 a1.channels.c1.capacity = 1000
 a1.channels.c1.transactionCapacity = 100

#Customizing sink for hdfs
a1.sinks.k1.hdfs.path = hdfs://quickstart.cloudera:8020/user/cloudera/virtualsensor1
a1.sinks.k1.hdfs.filePrefix = netcat
a1.sinks.k1.hdfs.fileType = DataStream

# Bind the source and sink to the channel
 a1.sources.r1.channels = c1
 a1.sinks.k1.channel = c1
```

Fig. 28 Forecasting sensor values using ARIMA model

```
Arima model forecast.R
Source on Save | Run | Source
1  # load sensor data
2  dataset = read.csv('Users/Riz/temperature_sensor.csv',
3                     check.names = FALSE, encoding = "UTF-8", blank.lines.skip = FALSE);
4
5  # load forecast library
6  library(forecast)
7
8  # convert sensor values to data frame
9  c1<- data.frame(dataset$Value)
10
11 # create an Arima model
12 fitmod <- Arima(c1, order=c(2,1,2), seasonal = TRUE)
13
14 # plot the next 1000 sensor values based on the arima model
15 plot(forecast(fitmod,h=1000), main = "Forecasting temperature from historical data",
16      xlab = "time", ylab = "sensor_values")
```

Funding Open Access funding enabled and organized by CAUL and its Member Institutions.

Declaration

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Cisco, the internet of things how the next evolution of the internet is changing everything. 2011.Available from https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf. Accessed 22 July 2022.
2. Al-Sai ZA, Abdullah R, Husin MH. Big data impacts and challenges: a review. In 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT). 2019.
3. Varma C. Performance analysis and challenges of the map reduce framework in big data analytics. In 2018 International Conference on Current Trends towards Converging Technologies (ICCTCT). 2018.
4. Jamil A, et al. Comprehensive review of challenges & technologies for big data analytics. In 2018 IEEE International Conference on Computer and Communication Engineering Technology (CCET). 2018.
5. Desai PV. A survey on big data applications and challenges. In 2018 Second International Conference on Inventive

Communication and Computational Technologies (ICICCT). 2018.
6. Caragliu A, Del Bo C, Nijkamp P. Smart cities in Europe. J Urban Technol. 2011;18(2):65–82.
7. Vasicek D, et al. IoT Smart Home Concept. In 2018 26th Telecommunications Forum (TELFOR). 2018.
8. Arnone D, et al. Energy management of multi-carrier smart buildings for integrating local renewable energy systems. In 2016 IEEE International Conference on Renewable Energy Research and Applications (ICRERA). 2016.
9. El-Shafie M, Fakeih L. The smart environment of commercial buildings. In 2018 15th Learning and Technology Conference (L&T). 2018.
10. Chen S-Y, et al. Survey on smart grid technology. Power Syst Technol. 2009;8:1–7.
11. Adeli H, Jiang X. Intelligent infrastructure: neural networks, wavelets, and chaos theory for intelligent transportation systems and smart structures. Boca Raton: Crc Press; 2009.
12. Demirkan H. A smart healthcare systems framework. It Professional. 2013;15(5):38–45.
13. Saha HN, et al. Disaster management using Internet of Things. In 2017 8th Annual Industrial Automation and Electromechanical Engineering Conference (IEMECON). 2017.
14. Dineshreddy V, Gangadharan GR. Towards an "Internet of Things" framework for financial services sector. In 2016 3rd International Conference on Recent Advances in Information Technology (RAIT). 2016.
15. Dlamini NN, Johnston K. The use, benefits and challenges of using the Internet of Things (IoT) in retail businesses: A literature review. In 2016 International Conference on Advances in Computing and Communication Engineering (ICACCE). 2016.
16. Chourabi H, et al. Understanding smart cities: an integrative framework. In System Science (HICSS), 2012 45th Hawaii International Conference on. IEEE. 2012.
17. Zhang K, et al. Security and privacy in smart city applications: challenges and solutions. IEEE Commun Mag. 2017;55(1):122–9.
18. Nath NR, Narayanaswami P, Mohan GG. Intelligent query placement strategy for progressive-real time analytics in big data. In 2016 10th International Conference on Intelligent Systems and Control (ISCO). 2016.
19. Ashamalla A, Beydoun G, Low G. Model driven approach for real-time requirement analysis of multi-agent systems. Comput Lang Syst Struct. 2017;50:127–39.
20. Miller RB. Response time in man-computer conversational transactions. In Proceedings of the December 9–11, 1968, fall joint computer conference, part I. 1968.
21. Bashir MR, et al. Big data management and analytics metamodel for IoT-enabled smart buildings. IEEE Access. 2020;8:169740–58.
22. Bashir MR, Gill AQ. Towards an IoT big data analytics framework: smart buildings systems. In 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS). IEEE. 2016.
23. Autoregressive integrated moving average. 2018. Available from: https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average. Accessed 2018 March 05.
24. Patel K, et al. Internet of things-IOT: definition, characteristics, architecture, enabling technologies, application & future challenges. In International journal of engineering sciencecomputing 2016.
25. Antón-Haro C, Dohler M. 1 - Introduction to machine-to-machine (M2M) communications. In: Machine-to-machine (M2M) communications. Oxford: Woodhead Publishing; 2015. p. 1–23.
26. Ericsson. More than 50 billion connected devices. 2011. Available from https://www.semanticscholar.org/paper/More-than-50-billion-connected-devices/c27a6e0d36e636b4904ad78e2136829602b3f150#extracted. Accessed 20 Aug 2020.
27. Mocrii D, Chen Y, Musilek PJIOT. IoT-based smart homes: a review of system architecture, software, communications, privacy and security. Internet of Things. 2018;1:81–98.
28. Akkaya K, et al. IoT-based occupancy monitoring techniques for energy-efficient smart buildings. In 2015 IEEE Wireless communications and networking conference workshops (WCNCW). IEEE. 2015.
29. Keertikumar M, Shubham M, Banakar R. Evolution of IoT in smart vehicles: an overview. In 2015 International Conference on Green Computing and Internet of Things (ICGCIoT). IEEE. 2015.
30. Sung WT, Lu CY. Smart warehouse management based on IoT architecture. In 2018 International Symposium on Computer, Consumer and Control (IS3C). IEEE. 2018.
31. Stavropoulos TG, et al. IoT wearable sensors and devices in elderly care: a literature review. Sensors. 2020;20(10):2826.
32. Talavera JM, et al. Review of IoT applications in agro-industrial and environmental fields. Comput Electron Agric. 2017;142:283–97.
33. Rehman HU, Asif M, Ahmad M. Future applications and research challenges of IOT. In 2017 International conference on information and communication technologies (ICICT). IEEE. 2017.
34. Uusitalo MA. global vision for the future wireless world from the WWRF. IEEE Veh Technol Mag. 2006;1(2):4–8.
35. Inamdar M, Roy S. Internet of things: architecture, security and applications. Int J Adv Eng Manag. 2017;2:157.
36. Joshi YA, et al. 3rd International Conference on Innovations in Automation and Mechatronics Engineering 2016, ICIAME 2016 05–06 February, 2016 Customized IoT enabled wireless sensing and monitoring platform for smart buildings. Procedia Technology. 2016;**23**: 256–263.
37. Bayerdörffer E, et al. Randomized, multicenter study: on-demand versus continuous maintenance treatment with esomeprazole in patients with non-erosive gastroesophageal reflux disease. BMC Gastroenterol. 2016;16:48.
38. Lin Z, Carley K. Proactive or reactive: an analysis of the effect of agent style on organizational decision-making performance. Intell Syst Account Financ Manag. 1993;2(4):271–87.
39. Dykes B. Reporting vs. analysis: what's the difference? 2010. Available from https://blog.adobe.com/en/publish/2010/10/19/reporting-vs-analysis-whats-the-difference. Accessed 15 Mar 2022.
40. Duffy A, O'Donnel FJ. A design research approach. In Proceedings of the AID'98 workshop on research methods. 1998. Lisbon, Portugal.
41. Shyam R, et al. Apache spark a big data analytics platform for smart grid. Procedia Technol. 2015;21:171–8.
42. Kolozali S, et al. A knowledge-based approach for real-time IoT data stream annotation and processing. In Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing(CPSCom), IEEE. 2014.
43. Moreno MV, et al. Big data: the key to energy efficiency in smart buildings. Soft Comput. 2016;20(5):1749–62.
44. Ismail A. Utilizing big data analytics as a solution for smart cities. In 2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC). 2016.
45. Plageras AP, et al. Efficient IoT-based sensor BIG Data collection–processing and analysis in smart buildings. Futur Gener Comput Syst. 2018;82:349–57.
46. Linder L, et al. Big building data—a big data platform for smart buildings. Energy Procedia. 2017;122:589–94.

47. Souza A, et al. Using big data and real-time analytics to support smart city initiatives. IFAC-PapersOnLine. 2016;49(30):257–62.
48. Rathore MM, Ahmad A, Paul A. IoT-based smart city development using big data analytical approach. In 2016 IEEE International Conference on Automatica (ICA-ACCA). 2016.
49. Ta-Shma P, et al. An ingestion and analytics architecture for IoT applied to smart city use cases. IEEE Internet Things J. 2017;5(2):765–74.
50. Strohbach M, et al. Towards a big data analytics framework for iot and smart city applications. In: Xhafa F, et al., editors. Modeling and processing for next-generation big-data technologies: with applications and case studies. Cham: Springer International Publishing; 2015. p. 257–82.
51. Constante-Nicolalde F-V, Pérez-Medina J-L, Guerra-Terán P. A proposed architecture for iot big data analysis in smart supply chain fields. Cham: Springer International Publishing; 2020.
52. Habibzadeh H, et al. Soft sensing in smart cities: handling 3Vs using recommender systems, machine intelligence, and data analytics. IEEE Commun Mag. 2018;56(2):78–86.
53. Kumar A, et al. Secure and energy-efficient smart building architecture with emerging technology IoT. Comput Commun. 2021;176:207–17.
54. Kumar M, et al. An efficient framework using visual recognition for IoT based smart city surveillance. Multimedia Tools and Applications. 2021;80(20):31277–95.
55. Kumar A, et al. Revolutionary strategies analysis and proposed system for future infrastructure in internet of things. Sustainability. 2022;14(1):71.
56. Kumar A, et al. Energy-efficient fog computing in internet of things based on routing protocol for low-power and Lossy Network with Contiki. Int J Commun Syst. 2022;35(4):e5049.
57. Li S, Xu LD, Zhao S. The internet of things: a survey. Inf Syst Front. 2015;17(2):243–59.
58. Al-Fuqaha A, et al. Internet of things: a survey on enabling technologies, protocols, and applications. IEEE Commun Surv Tutor. 2015;17(4):2347–76.
59. Chen F, et al. Data mining for the internet of things: literature review and challenges. International Journal of Distributed Sensor Networks. 2015;11(8):431047.
60. Gubbi J, et al. Internet of things (IoT): a vision, architectural elements, and future directions. Futur Gener Comput Syst. 2013;29(7):1645–60.
61. Marjani M, et al. Big IoT data analytics: architecture, opportunities, and open research challenges. IEEE Access. 2017;5:5247–61.
62. Prat N, Comyn-Wattiau I, Akoka J. Artifact evaluation in information systems design-science research—a holistic view. In Pacis Proceedings. 23. 2014.
63. Vaishnavi V, Kuechler B. Design science research in information systems. January 20, 2004 November 15, 2015; Available from: http://www.desrist.org/design-research-in-information-systems/.
64. Checkland P, Holwell S. Information, systems, and information systems. Chichester: John Wiley & Sons; 1998.
65. Hevner A, Chatterjee S. Design science research in information systems. In: Design research in information systems: theory and practice. Boston: Springer; 2010. p. 9–22.
66. Hevner AR. A three cycle view of design science research. Scand J Inf Syst. 2007;19(2):4.
67. Prat N, Comyn-Wattiau I, Akoka J. Artifact evaluation in information systems design-science research-a holistic view. In PACIS. 2014.
68. Gill AQ. Agile enterprise architecture modelling: evaluating the applicability and integration of six modelling standards. Inf Softw Technol. 2015;67:196–206.
69. Gill AQ, Adaptive cloud enterprise architecture. World Scientific. 2015. Available from https://www.worldscientific.com/worldscibooks/10.1142/9363#t=aboutBook. Accessed 25 May 2021.
70. Anwar MJ, Gill AQ. A review of the seven modelling approaches for digital ecosystem architecture. In 2019 IEEE 21st Conference on Business Informatics (CBI). 2019.
71. Anwar M, Gill A, Beydoun G. Using adaptive enterprise architecture framework for defining the adaptable identity ecosystem architecture. In Australasian Conference on Information Systems. 2019.
72. Microsoft. Microsoft Power BI. 2020. Available from: https://powerbi.microsoft.com/. Accessed 2015.
73. Tableau. Tableau. 2020. Available from: https://www.tableau.com/. Accessed 2020 December 10.
74. Apache flume. 2018. Available from: https://flume.apache.org/. Accessed 2018 August 10.
75. Apache Spark. Available from: http://spark.apache.org/. Accessed 2021 August 19.
76. Elasticsearch. 2018. Available from: https://www.elastic.co/products/elasticsearch. Accessed 2018 March 05.
77. Kibana. 2018. Available from: https://www.elastic.co/products/kibana. Accessed 2018 March 05.
78. Apache Lucene. 2018. Available from: http://lucene.apache.org/. Accessed 2018 July 07.
79. Logstash. 2018. Available from: https://www.elastic.co/products/logstash. Accessed 2018 August 10 .
80. Ahmed F. An IoT-big data based machine learning technique for forecasting water requirement in irrigation field. In International conference on research and practical issues of enterprise information systems. 2017. Springer.
81. Tableau. 2018. Available from: https://www.tableau.com/. Accessed 2018 March 05.
82. Weng T, Agarwal Y. From buildings to smart buildings—sensing and actuation to improve energy efficiency. IEEE Des Test Comput. 2012;29(4):36–44.
83. Cloudera. 2018 Available from: www.cloudera.com. Accessed 2018 March 05.
84. 5 Most important difference between apache kafka vs flume. 2020. Available from: https://www.educba.com/apache-kafka-vs-flume/. Accessed 19 July 2021.
85. Adeli H, Vishnubhotla PR. Parallel processing and parallel machines. In: Parallel processing in computational mechanics. Boca Raton p: CRC Press; 2020. p. 1–20.
86. Agarwal Y, et al. Occupancy-driven energy management for smart building automation. In Proceedings of the 2nd ACM workshop on embedded sensing systems for energy-efficiency in building. 2010.
87. Power BI. 2018. Available from: https://powerbi.microsoft.com/en-us/. Accessed 2018 March 05.
88. Apache Pig. Available from: https://pig.apache.org/. Accessed 2020 December 21.
89. Pycharm. 2018. Available from: https://www.jetbrains.com/pycharm/. Accessed 05 Mar 2018 .
90. Bashir R. smartBuildings. 2017. Available from: https://github.com/c3212218/smartBuildings. Accessed 20 July 2021.