The Challenges and Prerequisites of Data Stream Processing in Fog Environment for Digital Twin in Smart Industry

https://doi.org/10.3991/ijim.v15i15.24181

Ameer B. A. Alaasam(✉)
South Ural State University, Chelyabinsk, Russian Federation
alaasamab@susu.ru

Abstract—Smart industry systems are based on integrating historical and current data from sensors with physical and digital systems to control product states. For example, Digital Twin (DT) system predicts the future state of physical assets using live simulation and controls the current state through real-time feedback. These systems rely on the ability to process big data stream to provide real-time responses. For, example it is estimated that one autonomous vehicle (AV) could produce 30 terabytes of data per day. AV will not be on the road before using an effective way to managing its big data and solve latency challenges. Cloud computing failed in the latency challenge, while Fog computing addresses it by moving parts of the computations from the Cloud to the edge of the network near the asset to reduce the latency. This work studies the challenges in data stream processing for DT in a fog environment. The challenges include fog architecture, the necessity of loosely-coupling design, the used virtual machine versus container, the stateful versus stateless operations, the stream processing tools, and live migration between fog nodes. The work also proposes a fog computing architecture and provides a vision of the prerequisites to meet the challenges.

Keywords—stream processing, digital twin, fog computing

1 Introduction

Smart industry (or Industry 4.0) integrates the Internet of Things (IoT) with manufacturing techniques to create an interconnected manufacturing enterprise that analyzes the information to make intelligent action in the physical world [1]. An example of such integration is the digital twin (DT), which has gained extensive attention from researchers in the industry. DT contains three main components; the physical asset in real space and its virtual representation in virtual space, in addition to the data and information that integrate the real and virtual components [2].

Unlike the traditional simulation, the virtual representation in DT is continually updated with the state of maintenance and performance throughout the physical asset's life cycle [3]. For example, in car racing, the data stream from sensors on the car transmitted to the pit wall to simulate car performance and enable real-time remote adjustment [4]. The virtual representation model may be predefined or learned from the data streams, which is a complex process that may require machine learning and identification techniques to specify the parameters for each specific class of model [5][6]. Thus, DT's productivity is based on efficiently and securely transferring and analyzing real-time streaming data between physical assets and data processing systems [7]. Such a process requires a massive computing capability to manage data [8]. Cloud computing allows meeting such requirements for computing infrastructure [9].

Cloud computing enables dynamic resource sharing and provisioning by leveraging virtualization technologies at hardware and application levels [10]. The challenge is that cloud computing hardly meets the requirements of location-aware and delay-sensitive systems such as DT due to high latency [11], which is the motivation toward fog computing. Fog Computing is a virtualized platform that provides storage, compute and networking between the end physical asset and data center in the public cloud [12]. While the industrial data is often unstructured, it can be refined and preprocessed locally at the fog level before being sent to the cloud level for further processing [13]. The complexity of managing these challenges is the increase in DT, where there are multiple stateful streams.

Statefulness means that the system should always identify each data source and determine what other data produced by the same source over a time scale [14]. For example, the sequence of events so far encountered should be stored when the system searches for specific patterns over the data stream. The complexity of developing stateful systems such as DT is affected by the used processing tools to manage the state and the computing infrastructure's capabilities to keep the state of the process. These multiple challenges make finding appropriate solutions to handle data streams in a fog environment a critical issue, especially when building complex systems such as the DT.

This work provides an analytical review of the challenges of data stream processing in a fog environment to support DT implementation. The work also provides a vision of the prerequisites and proposes a conceptual fog architecture to meet the studied challenges. Section 2 explains the concept of DT, and section 3 provides an overview of the data stream processing and its requirements. Section 4 provides a study on the role of fog computing and its challenges in DT implementation. Based on the studied challenges, section 5 provide the required prerequisites and proposes a conceptual fog architecture. Conclusions are presented in section 6.

2 Digital twin

The wide availability of IoT sensors and the fast evolution of data gathering and processing tools allow the DT to become a reality in the market. In 2019 DT market size exceeded 4 billion USD and is estimated to grow at a CAGR of over 30% from 2020 to 2026 [15]. Figure 1 shows the result of Google trends analysis of worldwide search interest, which reflects the preference of "Digital Twin" versus "Industrial IoT" and IoT "Platform" for the past seven years from 2014 till 2021, where a value of 100 is the peak popularity and value of 0 means there was not enough data [16].

In 2002 Dr. Michael Grieves presented the first concept of DT at the University of Michigan to establish a Product Lifecycle Management center (PLM) [8]. The idea was that digital information constructed from the physical asset would be a "twin" of the information embedded in the physical asset itself and together are linked through the system's entire life cycle. DT is an integrated multiscale, multiphysics, probabilistic simulation of a product that uses the best available physical models, sensor updates, system history to mirror the life of its physical twin [17].

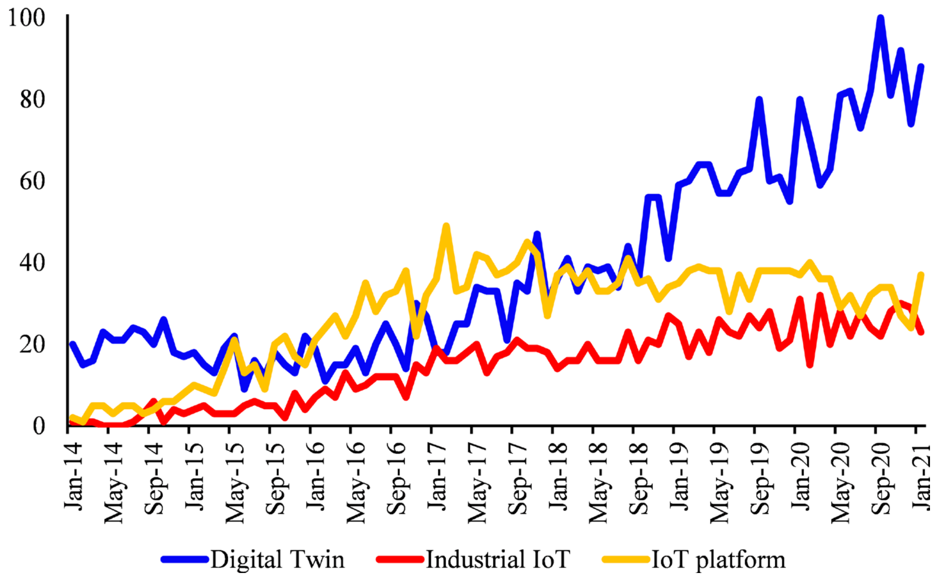

Fig. 1. Google trends analysis of worldwide search interest in "Digital Twin" versus "Industrial IoT" and "IoT Platform" for the period from 2014 till 2021

3 The role of the data stream

3.1 Stream definitions

The stream of data is a countably infinite sequence of elements that represent data elements that are made available over time scale, for example, readings from sensors, stock quotes in financial applications [18]. Stream processing means to perform various analysis tasks on the stream rather than batch fashion [19]. These tasks may consist of building models to create a predictor or discover frequent patterns. Other examples are credit card fraud detection in online transactions [20], and anonymity methods to masking the message meta content that identifies the senders and receivers [21].

According to [22], stream processing algorithms operate sequentially over unbounded input streams and produce output streams as an answer over the event observed so far or a sliding window of recent data to answer continuous queries. Stream processing algorithms can be categorized according to (1) the type of output which the algorithm computes (e.g., is the answer approximate or exact), (2) how the algorithm computes the output (e.g., hashing, sampling).

3.2 Stateless and stateful operations

The operations in the stream processing algorithm can be classified into stateless and stateful. A stateless operation transforms each input data point one at a time and outputs the result based solely on that last input. In contrast, a stateful operation maintains the value of data points processed so far. It updates the value with each new input, such that the output reflects results that take into account both the new and historical inputs [23]. The stateful operation has many challenges, for example, scalability limitations and the need for additional storage and computing service to support state management [24]. For example, when multiple instances of the same operation at the same time trying to process the same input that required the same historical state, in this case, there are no guarantees for the correctness of execution; thus, the state must be processed by one instance [5]. Managing the state are addressed by different methods in the set of available data stream platforms as in the following:

1. Kafka streaming platform [25] provides so-called state stores within the applications to manage the state; for example, the application developed using Kafka Streams DSL API automatically creates and manages state stores when stateful operations are called. Kafka Streams provide automatic recovery for local state stores by synchronizing the local state with the Kafka messaging middleware itself, which was the source of input data and the destination of output result.
2. Flink, which is a distributed processing engine for stateful computations over data streams, manages the state by creating local persisted states inside the application, but for fault tolerance, Flink provides snapshotting mechanism with an external resource such as HDFS and S3 [26].
3. Spark [27] is an analytics engine for large-scale data processing. Spark also provides a stateful capability where the data stream persisted in memory when a stateful operation is in use, such as a window-based operation, but for fault-tolerant spark needs to checkpoint information to a fault-tolerant storage system such as HDFS.

There are architectural differences between Kafka Stream API on the one hand and Flink [28] and Spark on the other hand. Table 1. shows the interesting differences between Kafka Stream API, Spark, and Flink based on the literature analysis.

3.3 Toward data stream in digital twin

The complexity is increasing dramatically in complex systems such as DT, where there are multiple stateful streams between real-world objects, fog nodes, and the cloud. Thus, managing data stream processing in the smart industry system such as DT gained extensive attention from researchers. Authors in [29] developed a methodology to capture real-time sensor data from cyber-physical systems communicating the factory's physical assets' current status to the real-time DT model. The authors in [30] presented a proof of concept for DT where the DT components connected through a broker-client-architecture followed the publish-subscribe model implemented using the MQTT protocol. The authors of [31] propose developing DT using the APIs mediation layer, Inner APIs connected directly to the DT, and Outer APIs are exposed through mediation which can be composite endpoints by joining two or more inner APIs. The mediation layer allows maintaining the experience of integration through an additional layer of management. The authors of [32] propose a microservice-based middleware that offers an API gateway for managing DT. The DT, where the platform implements a microservice architecture, supports the distributed publication of simulation models and manages data stream coming from the shop-floor for physical-digital synchronization.

Table 1. Comparison between stateful stream processing frameworks: Kafka Stream API, Flink, and Spark based on the literature analysis

Criteria	Kafka Streams API	Flink and Spark
Deployment	API that can be embedded in the application and did not impose a specific deployment method	The framework deploying the application, either in standalone clusters, or using YARN, Mesos, or containers
Stateful and fault-tolerant	State store locally and synchronized with the native Kafka message queue	State locally but for fault-tolerant can be configured with external storage such as HDFS
Source of streaming data	Only from Kafka messages queue that supports the Connect API, producer API, and Consumer API in Kafka to address the problem of data in/out from another system	Kafka, File Systems, other message queues
Sink for results	Kafka, application state, database, or any external system	Kafka, other message queues, file system, and other external systems
Bounded and unbounded data streams	Unbounded	Unbounded and Bounded

4 The role of fog computing

One of the contemporary revolutionary products in the smart industry; is the autonomous vehicle (AV). The global market of AV is expected to reach 52.4 billion USD in 2027 at an impressive 14.5% CAGR during the forecast period from 2021 to 2027 [33]. While Twitter, with 270 million users in 2018, produces about 100 GB of data per day, but it was estimated that a single AV vehicle could produce 30 terabytes of data in a single day of driving [34]. Without an effective way to managing all this data, AV will not be on the road before encountering bandwidth and latency challenges. Another example is the real-time electrocardiogram systems (ECG) that extract features from the patients ECG signals, where avoiding high latency is essential to rescue human life [35]. High latency is the challenge that makes Cloud computing fail in meeting the delay-sensitive and location-aware systems [11]. That was the motivation toward Fog computing, a virtualization platform that provides compute, storage, and networking services between the end physical asset and Cloud data center [12].

4.1 Virtualization and statefulness challenge

Further complexity is faced in fog computing when the live migration of storage and processing tasks between fog nodes is required. For example, the Hyper-V feature in Windows Server can transparently migrate running Virtual Machines (VM) from one host to another without perceived downtime [36]. But, using VM is not adaptable to fog computing aims; for example, the boot-up time of a VM is several minutes, which is too long for real-time applications; also, the physical nodes' performance is degraded when the number of VMs increased [11]. Nevertheless, large overheads associated with the use of VMs can limit the efficiency of computational resources, especially at the fog level where there is limited computational power in the edge devices, and the bandwidth is typically limited in the edge of the network. Container-based virtualization can address this problem.

The container-based virtualization does not aim to emulate the entire hardware environment as it in hypervisor-based virtualization. Instead, it enables the OS kernel to isolate the applications where multiple isolated OS systems (containers) run on a host and sharing a single kernel instance [37]. Container-based virtualization makes it easy to package the application with its dependencies into a small container with lower overhead than VM [38]. Nevertheless, another challenge arises in containers where it is challenging to containerize a stateful operation due to limited support for state portability in containers. Docker, for example, using the commit [39] command, can snapshot a running container; however, this operation only saves the container file changes and settings into a new docker image regardless of the state of the running processes.

To tackle the challenge of container live migration, the authors of [40] proposed the Voyager framework to migrate the container state across three different data stores: in-memory, local filesystem, and network filesystem. Local filesystem migration in Voyager starts with the data federation step in which the container data at the source host become accessible on the target host using the NFS server. When the container in the source host resumed access to its data using data federation, Voyager launches a lazy replicator to copy and transfer the data from the source to the target source. For any network-attached file storage and host access authorization Voyager performs unmounting and mounting through the NAS share server. For in-memory state migration Voyager used the CRIU project. However, the CRIU project [41] is still based on Docker's experimental mode until the writing time. Also, checkpointing and restoring the container state using CRIU still does not support all storage drivers for container file system management in Docker [42]. Thus, it essential to develop efficient solutions for the challenges of Statefulness in the used underline virtualization techniques, especially when live migration is required.

4.2 Fog architecture and computing infrastructure

The authors of [43] introduced a hierarchical 4-layers fog architecture for big data analysis in smart cities to support quick response at neighborhood-wide, community-wide, and city-wide levels. The hierarchical 4-layers fog architecture proposed by authors contains three levels; infrastructure level (physical assets), fog level, and cloud level. The fog level contains three layers: 1) The layer of fiber optic sensing networks responsible for the tasks that may require a millisecond, 2) the layer of edge computing nodes to perform further tasks that may require seconds, and it aims to reduce the load between the edge devices and the upper layer, 3) the layer of intermediate computing nodes responsible for the tasks that may require minutes or hours. Finally, at the cloud level, the cloud data center layer is existing and is responsible for tasks that may require days or years. The authors in [44] proposed an SDN-based architecture where the SDN is placed in the middle of the fog node and the cloud to improve communication efficiency between the fog node and the cloud. They also proposed a service architecture that includes two layers: the user layer and the application layer. The users private VM is deployed at the user layer, where the user data processed and sent to the service VM at the application layer.

Based on the open-source Kura gateway, the authors of [45] implemented a fog-oriented framework to run IoT applications delivering containerized applications. The authors used resource-limited Fog Node, like Raspberry Pi, and conducted the tests to evaluate the existing solutions, e.g., Kubernetes, Apache Mesos, and Docker Swarm; they decided to use Docker Swarm as they find it more lightweight than others. The authors of [11] proposed container-based fog computing architecture to reduce the service delay and improve fog nodes resource utilization. The authors proposed that a fog node consists of three tiers; 1) infrastructure tier, 2) the control tier, and 3) the access tier. The request received by the fog node will be validated and resolved into the service catalog at the access tier and transferred to the control tier. According to the service type, the control tier dispatches the services to the corresponding manager, either long-term or temporary. Finally, the manager calls the API provided by the infrastructure tier to create the container.

5 The proposed architecture and prerequisites

Based on the analytic study which carried on state of the art in the area of stream processing, fog computing, and DT in the smart industry, the current work in the following subsections can provide vision on the required architecture and the prerequisites for the implementation of DT in the smart industry within fog environment.

5.1 Loosely coupled architecture

The system should be refactored as a set of loosely coupled components. Each component can perform a specific task built around a specific business logic and can respond to events independently. This design provides the ability to migrate parts of the processing to different nodes as required in the fog paradigm and provide a faster response to events separately. This design also provides independent deployments for each system component according to each component’s specific needs.

5.2 Stateful virtualization infrastructure

The complexity of designing the stateful system is affected by the capabilities of the underlying computing infrastructure. According to [5], stateful virtualization infrastructure is defined as the infrastructure that allows storing and managing the data internally over a time scale and under a variety of influential factors and use cases. When the infrastructure is suffering from less Statefulness capability, additional efforts are required to manage stateful components in the specific circumstances when the data loss problem can arise, such as in live migration.

5.3 Processing and data portability

When the physical asset is moving across different geographic locations, there is a need to access the nearest data center for high performance and low latency. Also, when a data center disaster, the failing computing units can use a different data center, which includes a replication of the original data. Another scenario, when data need to be aggregated from different data centers which may in different locations. These scenarios demand the need to provide the portability of both data repositories and processing to be available in the required geographical location and access them easily. Processing portability and data portability have different characteristics.

Processing portability means the distance between the processing service and physical assets affects latency and performance; thus, it is imperative to provide portability for the relevant processing service to be in the required location that may change over time. There are several challenges in processing portability. Two classes of these challenges can be provided here: the re-installation challenge and the re-setting challenge:

1. In the *re-installation challenge*, installing an application on one host and then reinstalling it again at another host, with all its dependencies and configurations, does not provide enough flexibility to be considered portable. Even with some applications that do not need to be installed, they also face challenges when changing the host, such as OS incompatibility. Containerization arises as a lightweight solution to this problem. For example, in Docker, the application with all its dependencies packaged in a container and commit the changes to an image. The Docker image can be downloaded in any host with Docker, then rerun the pre-configured container regardless of the host OS.
2. The *re-settings challenge* appears, for example, when the host is changed, which means that some settings may need to update, such as the new IP. In this case, from the development stage, the environmental parameters that are subject to change should be as variables that can be re-setting without changing the application. Many deployment methods support solutions for this problem; for example, in Docker, when the new container is launched, the new settings can be passed as parameters in the running time.

Data portability means there is a need to manage data in data layers isolated from processing layers that contain the processing services. Data layers should provide straightforward data access and delivery for each processing service in processing layers and replicate data across different nodes in different locations. For example, to provide fault-tolerant, Spark and Flink use external storage such as HDFS, while Kafka Stream DSL API synchronizes the application state with Kafka message queue.

Separate data layers allow avoiding data loss in many scenarios, for example, when a processing service fails caused by a disaster in the computing node. Another example, when physical asset moves to a different location, a replication or migration of data for the nearest data node can provide a solution for a high latency challenge.

In the fog environment, the computing is distributed as independent components across different implementation platforms, there is always a data stream from the main sources, stream input to components, intermediate status data for each component, and stream output from each component. In this case, the streaming middleware as a data layer can take the role of nervous systems for the data sources and the independent processing services. Figure 2 shows the proposed fog computing architecture with data layers and processing layers to support the implementation of DT.

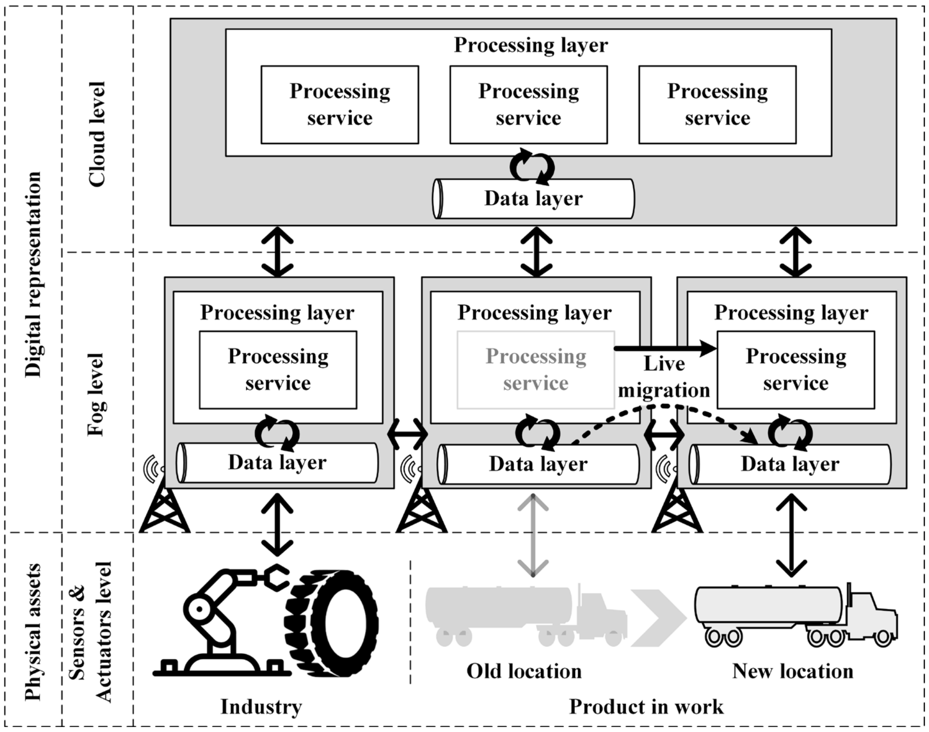

Fig. 2. The proposed fog computing architecture to support the implementation of DT

5.4 Computing stages

Big industrial IoT data is often unstructured; it can be preprocessed, refined, and monitored locally or in a fog node before being sent to the cloud for further processing. However, we can define four sequential computing stages regardless of the implementation place. There are many visions of applying these 4-stage depending on the distributing approach and the computing design. Each stage may be applied separately in different computing components or one component containing more than one stage, or even all the stages applied in one component designed around the specific business logic. The 4-stage as in the following:

1. **Stage 1**: connected with the data source and contains the initial steps applied to the raw data such as data extraction, validation, cleaning, etc.
2. **Stage 2**: which applied to the refined data and it may include data integrating, aggregating, windowing, etc.
3. **Stage 3**: includes applying the required business logic and analytics to the incoming data from Stage 2. The schema and behavior of the result of this stage are often different from the incoming data. For example, the input may be a lot of aggregated messages, including a timestamp and other information, while the result may only be a message with true or false.
4. **Stage 4**: it includes archiving or publishing the results for further processing.

5.5 Data and processing integration

When both data layers and the processing layers are isolated and distributed over various implementation platforms at different levels, there is a need for an integration mechanism that maps data between these layers. Integration can be illustrated on two levels as follows:

1. **Integration at the data level**: In a complex system such as DT, there are many data sources. For example, a heterogeneous group of sensors or even the processing services themselves produce data as well. Each specific group of these sources may generate data from different physical locations. Therefore, in such a highly distributed environment, data that often being processed together based on the required business logic, or data from similar sources, it is better to be organized in a unified structure and accessed through similar interfaces.
2. **Integration at the processing level**: When the tight links between the different processing components are decoupled, there is a need for an integration mechanism that allows maps the result from one processing component to another that needs those results. Also, the processing components that are assumed to be cooperated to solve a specific problem should have the same ability to access the same data interface.

6 Conclusions

Fog computing is emerging to solve the high latency problem when using cloud computing alone to implement real-time systems such as DT. In fog computing, a set of the processing services moving to be near the physical assets. Thus, the work identifies the need to refactor the system as a set of loosely coupled components; each performs a specific task built around a specific business logic and can respond to events independently. Each component can be migrated to different nodes as required in the fog paradigm and provide a faster response to events separately and independent deployments for each component. However, going towards fog computing puts many challenges and requirements at the fore.

According to the study that was conducted in this work, many of the challenges were listed along with the solutions that were proposed. For example, container technology appears as a solution to high costs when using VM. However, we face losing the application state in containers due to less data portability than VM. The problem increased when live migration of computation is required between fog nodes. Yet, this problem is not completely solved and needs further research and experiments to provide statefulness capability for containers. In terms of stateful streams processing frameworks such as Kafka, Flink, and Spark can provide a facility for stateful computation. However, each comes with different capabilities that should be taken into account when deployed stateful operations.

The need for data layers isolated from processing layers also identified by the current work. Data layers should provide data access and delivery for processing layers and replicate data across different nodes in different locations. The study also proposes the need for data portability and the need for processing portability in Fog environments by means of providing stateful computing and data in any location where it is needed. However, the study also identifies the need for integration mechanisms at the data level and computing level to ensure the correctness of data and processing workflows when the system is highly distributed. The work also proposes a conceptual fog architecture to meet the studied challenges and requirements.

However, many experiments must be conducted to address the studied challenges to provide a fog environment suitable to implement DT and an efficient stateful stream processing based on the prerequisites presented in this work.

7 Acknowledgment

The reported study was funded by RFBR, project number 19-37-90073.

8 References

[1] Parrott, A., & Lane, W. (2017). Industry 4.0 and the digital twin: Manufacturing meets its match. Deloitte University Press. 1–17. https://dupress.deloitte.com/dup-us-en/focus/industry-4-0/digital-twin-technology-smart-factory.html

[2] Grieves, M. (2014). Digital Twin: Manufacturing Excellence through Virtual Factory Replication. 1–7. https://theengineer.markallengroup.com/production/content/uploads/2014/12/Digital_Twin_White_Paper_Dr_Grieves.pdf

[3] Madni, A., Madni, C., & Lucero, S. (2019). Leveraging Digital Twin Technology in Model-Based Systems Engineering. Systems, 7(1): 7. https://doi.org/10.3390/systems7010007

[4] Modoni, G. E., Sacco, M., & Terkaj, W. (2016). A Telemetry-driven Approach to Simulate Data-intensive Manufacturing Processes. Procedia CIRP. 57: 281–285. https://doi.org/10.1016/j.procir.2016.11.049

[5] Alaasam, A. B. A., Radchenko, G., Tchernykh, A., & Gonzalez Compean, J. L. (2020). Analytic Study of Containerizing Stateful Stream Processing as Microservice to Support Digital Twins in Fog Computing. Programming and Computer Software. 46(8): 511–525. https://doi.org/10.1134/S0361768820080083

[6] Qamsane, Y., Chen, C. Y., Balta, E. C., Kao, B. C., Mohan, S., Moyne, J., Tilbury, D., & Barton, K. (2019). A unified digital twin framework for real-time monitoring and evaluation of smart manufacturing systems. In 2019 IEEE International Conference on Automation Science and Engineering. 1394–1401. https://doi.org/10.1109/COASE.2019.8843269

[7] Singh, S., Angrish, A., Barkley, J., Starly, B., Lee, Y.-S., & Cohen, P. (2017). Streaming Machine Generated Data to Enable a Third-Party Ecosystem of Digital Manufacturing Apps. Procedia Manufacturing. 10: 1020–1030. https://doi.org/10.1016/j.promfg.2017.07.093

[8] Grieves, M., & Vickers, J. (2017). Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. In: Kahlen FJ., Flumerfelt S., Alves A. (eds) Transdisciplinary Perspectives on Complex Systems. 85–113. https://doi.org/10.1007/978-3-319-38756-7_4

[9] Radchenko, G., Alaasam, A. B. A., & Tchernykh, A. (2018). Micro-Workflows: Kafka and Kepler Fusion to Support Digital Twins of Industrial Processes. In 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion). 18: 83–88. https://doi.org/10.1109/UCC-Companion.2018.00039

[10] Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications. 1(1): 7–18. https://doi.org/10.1007/s13174-010-0007-6

[11] Luo, J., Yin, L., Hu, J., Wang, C., Liu, X., Fan, X., & Luo, H. (2019). Container-based fog computing architecture and energy-balancing scheduling algorithm for energy IoT. Future Generation Computer Systems. 97: 50–60. https://doi.org/10.1016/j.future.2018.12.063

[12] Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog computing and its role in the internet of things. MCC'12 - Proceedings of the 1st ACM Mobile Cloud Computing Workshop. 13–15. https://doi.org/10.1145/2342509.2342513

[13] Aazam, M., Zeadally, S., & Harras, K. A. (2018). Deploying Fog Computing in Industrial Internet of Things and Industry 4.0. IEEE Transactions on Industrial Informatics. 14(10): 4674–4682. https://doi.org/10.1109/TII.2018.2855198

[14] Peiffer, C., & L'Heureux, I. (2013). System and method for maintaining statefulness during client-server interactions. (12) United States Patent US8346848B2. https://patentimages.storage.googleapis.com/85/8c/a9/845d051ef38264/US8346848.pdf

[15] Wadhwani, P., & Kasnale, S. (2020). "Digital Twin Market Statistics | Global Size Forecasts 2026," https://www.gminsights.com/industry-analysis/digital-twin-market (accessed Mar. 13, 2021)

[16] "Digital Twin, Industrial IoT, IoT platform—Explore—Google Trends." https://trends.google.com/trends/explore?date=2014-01-01 2021-01-01&q=Digital Twin,Industrial IoT,IoT platform (accessed Apr. 03, 2021).

[17] Glaessgen, E. H., & Stargel, D. D. S. (2012). The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles. In 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference—Special Session on the Digital Twin. 1–14. https://doi.org/10.2514/6.2012-1818

[18] Margara, A., & Rabl, T. (2019). Definition of Data Streams. In Encyclopedia of Big Data Technologies. 648–652. https://doi.org/10.1007/978-3-319-63962-8_188-1

[19] Gavalda, R. (2019). Adaptive Windowing. In Encyclopedia of Big Data Technologies. 1–6. https://doi.org/10.1007/978-3-319-63962-8_194-1

[20] Hussein, A. S., Khairy, R. S., Najeeb, S. M. M., & Alrikabi, H. Th. S. (2021). Credit Card Fraud Detection Using Fuzzy Rough Nearest Neighbor and Sequential Minimal Optimization with Logistic Regression. International Journal of Interactive Mobile Technologies (iJIM). 15(5): 24–42. http://dx.doi.org/10.3991/ijim.v15i05.17173

[21] Naman, H. A., Hussien, N. A., Al-dabag, M. L., & Alrikabi, H. Th. S. (2021). Encryption System for Hiding Information Based on Internet of Things. International Journal of Interactive Mobile Technologies (iJIM). 15(2): 172–183. https://doi.org/10.3991/ijim.v15i02.19869

[22] Golab, L. (2019). Types of Stream Processing Algorithms. In Encyclopedia of Big Data Technologies. 1726–1732. https://doi.org/10.1007/978-3-319-63962-8_193-1

[23] Liu, P., Xu, H., Da Silva, D., Wang, Q., Ahmed, S. T., & Hu, L. (2020). FP4S: Fragment-based Parallel State Recovery for Stateful Stream Applications. In 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS). 1102–1111. https://doi.org/10.1109/IPDPS47924.2020.00116

[24] Naseri, M., & Towhidi A. (2007). Stateful Web Services: A Missing Point in Web Service Standards. In Proceedings of the International MultiConference of Engineers and Computer Scientists 2007 (IMECS 2007). 993–997.

[25] "Streams Architecture—Confluent Documentation." https://docs.confluent.io/platform/current/streams/architecture.html (accessed Mar. 14, 2021).

[26] "Apache Flink Stateful Functions 2.2 Documentation: Application Building Blocks." https://ci.apache.org/projects/flink/flink-statefun-docs-stable/concepts/application-building-blocks.html#persisted-states (accessed Mar. 14, 2021).

[27] "Spark Streaming—Spark 3.1.1 Documentation." https://spark.apache.org/docs/latest/streaming-programming-guide.html#caching--persistence (accessed Mar. 14, 2021).

[28] "Flink vs Kafka Streams—Comparing Features." https://www.confluent.io/blog/apache-flink-apache-kafka-streams-comparison-guideline-users/ (accessed Mar. 14, 2021).

[29] Danilczyk, W., Sun, Y., & He, H. (2019). ANGEL: An Intelligent Digital Twin Framework for Microgrid Security. In 2019 North American Power Symposium (NAPS). 1–6. https://doi.org/10.1109/NAPS46351.2019.9000371

[30] Haag, S., & Anderl, R. (2018). Digital twin—Proof of concept. Manufacturing Letters. 15(B): 64–66. https://doi.org/10.1016/j.mfglet.2018.02.006

[31] Scheibmeir, J., & Malaiya, Y. (2019). An API Development Model for Digital Twins. In 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C). 518–519. https://doi.org/10.1109/QRS-C.2019.00103

[32] Ciavotta, M., Alge, M., Menato, S., Rovere, D., & Pedrazzoli, P. (2017). A Microservice-based Middleware for the Digital Factory. Procedia Manufacturing. 11:931–938. https://doi.org/10.1016/j.promfg.2017.07.197

[33] "Global Autonomous Vehicle Market- Industry Trends & Forecast Report 2027." https://www.blueweaveconsulting.com/global-autonomous-vehicles-market (accessed Mar. 14, 2021).

[34] "Driverless Car Data Storage | Automakers, Suppliers Grapple Overflow | WardsAuto." https://www.wardsauto.com/technology/storage-almost-full-driverless-cars-create-data-crunch (accessed Mar. 14, 2021).

[35] Al-dabag, M. L., ALRikabi, H. Th. S, & Al-Nima, R. R. O. (2021). Anticipating Atrial Fibrillation Signal Using Efficient Algorithm. International Journal of Online and Biomedical Engineering (iJOE). 17(2): 106–120. https://doi.org/10.3991/ijoe.v17i02.19183

[36] "Live Migration Overview | Microsoft Docs." https://docs.microsoft.com/en-us/windows-server/virtualization/hyper-v/manage/live-migration-overview (accessed Dec. 23, 2019).

[37] Li, W., & Kanso, A. (2015). Comparing containers versus virtual machines for achieving high availability. In proceedings—2015 IEEE International Conference on Cloud Engineering (IC2E 2015). 353–358. https://doi.org/10.1109/IC2E.2015.79

[38] Scheepers, M. J. (2014). Virtualization and Containerization of Application Infrastructure : A Comparison. In 21st Twente Student Conference on IT. 1–7.

[39] "docker commit | Docker Documentation." https://docs.docker.com/engine/reference/commandline/commit/ (accessed Mar. 21, 2021).

[40] Nadgowda, S., Suneja, S., Bila, N., & Isci, C. (2017). Voyager: Complete Container State Migration. In proceedings—International Conference on Distributed Computing Systems. (III): 2137–2142. https://doi.org/10.1109/ICDCS.2017.91
[41] "Docker—CRIU." https://criu.org/Docker (accessed: Mar. 21, 2021).
[42] "Docker External—CRIU." https://criu.org/Docker_External (accessed Mar. 21, 2021).
[43] Tang, B., Chen, Z., Hefferman, G., Wei, T., He, H., & Yang, Q. (2015). A Hierarchical Distributed Fog Computing Architecture for Big Data Analysis in Smart Cities. In Proceedings of the ASE BigData & SocialInformatics 2015 (ASE BD&SI 15). 1–6.
[44] Sun, X., & Ansari, N. (2016). EdgeIoT: Mobile Edge Computing for the Internet of Things. IEEE Communications Magazine. 54(12): 22–29. https://doi.org/10.1109/MCOM.2016.1600492CM
[45] Bellavista, P., & Zanni, A. (2017). Feasibility of Fog Computing Deployment based on Docker Containerization over RaspberryPi. In Proceedings of the 18th International Conference on Distributed Computing and Networking. 1–10. https://doi.org/10.1145/3007748.3007777

9 Author

Ameer B. A. Alaasam, assistant lecturer and PhD student at the School of Electronic Engineering and Computer Science, South Ural State University, Chelyabinsk, Russian Federation. The area of his scientific interests includes technologies of distributed computing systems, including methods of data stream processing, cloud and fog computing, workflow systems. https://orcid.org/0000-0002-2084-8899

Article submitted 2021-05-23. Resubmitted 2021-06-10. Final acceptance 2021-06-11. Final version published as submitted by the authors.