Blue Danube: A Large-Scale, End-to-End Synchronous, Distributed Data Stream Processing Architecture for Time-Sensitive Applications

Panayiotis A. Michael
University of California Los Angeles,
National Technical University of Athens
Athens, Greece
pm@ucla.edu

Panayiotis D. Tsanakas
Electrical Engineering/Computer Engineering Department
National Technical University of Athens
Athens, Greece
panag@cs.ntua.gr

D. S. Parker
Computer Science Department
University of California Los Angeles
Los Angeles, California
stott@cs.ucla.edu

***Abstract*—An extensive list of time-sensitive applications requiring ultra-low latency ranging from a few microseconds to a few milliseconds are presented in recent publications and IEEE standards. Time-sensitive applications, include industrial, critical healthcare and transportation applications as also applications for Smart Grids and the Internet of Vehicles -- one of the most active research fields of Intelligent Transportation Systems of Smart Cities. In this work, we mainly set our focus on the suite of safety applications which attracts strong interest from the research community, as it aims to avoid road accidents and save lives.**

The IEEE Time-Sensitive Networking (TSN) set of standards specifies fundamental real-time characteristics. Nevertheless, as TSN works on Data Link layer (Layer 2 of the OSI model) the benefits of these characteristics fade away when other layers are crossed from the Application layer (Layer 7). Indicatively, recent research works report latencies on the order of tens of seconds when benchmarking Data Stream Processing and IoT platforms, and thus they are not suited for time-critical applications. Such platforms mainly use loosely coupled components with asynchronous communication.

On Application layer, we propose a novel End-to-End Synchronous, Distributed Architecture for Large-Scale, High-Bandwidth, Ultra-Low Latency Data Stream Processing. Through our Big Data Stream analysis experiments (4.7 Gbit/s total average aggregated throughput, 1 Terabyte in-memory distributed database, 4 milliseconds average query latency) we have demonstrated the suitability of our architecture for time-sensitive applications such as accident avoidance for the Internet of Vehicles.

Keywords*—synchronous, data stream processing, end-to-end synchronous, time-sensitive applications, time-critical applications, ultra-low latency, Internet-of-Things, Intelligent Transportation Systems, Internet of Vehicles, Smart Cities

I. Introduction

A large number of current and future applications demand ultra-low end-to-end latency, a fact extensively presented in [1]. A rich list of time-sensitive application domains are referenced in the same publication, such as: industrial applications, tactile Internet, critical healthcare applications and transportation applications. The demands for ultra-low latency range from a few microseconds to a few milliseconds. Indicatively, for traffic efficiency and safety, QoS requirements reported in [1] necessitate latency to be below a 5ms upper bound, for industrial automation to be within a range between 0.2µs to 0.5ms and for power grid systems to remain at approximately 8ms. Similar ultra-low latency requirements are desired for Latency Critical Internet-of-Things applications as reported in [4]. In its Latency Critical IoT Use Cases, the publication includes Smart Grids and Intelligent Transportation Systems. The necessity for ultra-low latency while processing advanced analytics over very high bandwidth data streams prevails in the Internet of Vehicles (IoV) [8], one of the most active research fields in Intelligent Transportation Systems of Smart Cities. The IoV domain includes a suite of safety applications, the main objective of which is to avoid and reduce road accidents.

The Time-Sensitive Networking (TSN) set of standards specified by IEEE 802 [3] aims to provide deterministic connectivity through IEEE 802 networks. Time-Sensitive Networking standards -- IEEE 802.1, work at OSI Layer 2 guaranteeing packet transport with bounded latency, low packet delay variation, and low packet loss. Service Level Agreements (SLAs) are structured based on the real-time characteristics of the network. However, when considering end-to-end analysis, the ultra-low latency benefits gained through conformance to the TSN standards at Layer 2 can fade away when crossing different layers of the OSI model. For this reason, it's very important when performing end-to-end real-time analysis, to explicitly state on which layer we position each one of the two end-points of such analysis.

More specifically, in order to provide guarantees for real-time performance in terms of latency for an application, we need to cross all 7 layers of the OSI model starting at Layer 1 (Physical Layer) and ending at Layer 7 (Application Layer). This means that if the application is not well designed to exploit the real-time gains passed from the lower layers (e.g. from Layer 2 when conforming to the TSN standard), these gains can be diminished. The situation becomes more complex and more difficult to address in a cloud environment due to the existence of additional layers. Particularly, the authors of [7] propose a cloud stack analogous to the OSI model. The cloud stack develops on top of the infrastructure of a data center consisting of the following layers: Infrastructure (cloud stack Layer 1) which provides the foundation for the remainder of the stack, Hypervisor (cloud Layer 2), Software-Defined Data Center (cloud Layer 3), Image (cloud Layer 4), Services (cloud Layer 5) and Application (cloud Layer 6). It is expected for these additional cloud layers to introduce more complexity and cumulative increases of latency. We thus realize that guaranteeing ultra-low latency to a specific application in a large scale distributed environment (cluster or cloud), can be very challenging when crossing multiple layers in an end-to-end manner.

These challenges are more immense in data stream processing solutions. More specifically, by examining the benchmarking results of Apache Storm, Apache Spark and Apache Flink in [9] when using data from an online video game application, we see that event time latencies (end-to-end latencies) for all three systems were on the order of seconds. Such performance, cannot provide the guarantees expected by time-sensitive applications requiring latencies on the order of a few microseconds or a few milliseconds. In the same publication [9], it is explained that the latencies included various forms of delays, such as delays until windows were formed completely before being processed, delays due to data re-partitioning at the message broker and delays due to queues at data generators. Long end-to-end latencies between 60-90 seconds were also observed in [10] when benchmarking Microsoft Azure IoT Edge, due to the fact that Azure IoT service batches messages from an edge device in the IoT Hub in the cloud and it writes the results from the multiple messages as a batch in a single blob file.

As seen from the state-of-the-art, there is a significant gap to be filled when requesting end-to-end latencies on the order of microsecond or millisecond at the Application Layer. By examining the paradigm established by the standardization efforts of TSN on Layer 2, we realized that when working on the Application layer, in order to best exploit the real-time performance gains from the underlying network, we need to perform end-to-end optimization in order to achieve ultra-low latency across all layers.

A. *An alternative, novel, synchronous approach at Application level for real-time data stream processing*

In this work we perform an optimization process along the three dimensions of end-to-end latency, throughput and level of sophistication of the data analysis performed. The delays reported in [9] are inherent to the general approach followed when data stream processing is performed in distributed computing with loosely coupled components using asynchronous communication. We recognize that while there are many desirable properties of using loosely coupled components with asynchronous communication in distributed computing (including responsiveness, resilience and elasticity [11]), for time-sensitive applications -- especially within the context of real-time data stream processing, frameworks following a **synchronous communication paradigm** may be more appropriate.

By following an optimization process over the three dimensions of latency (minimization), bandwidth (maximization) and sophistication of advanced analytics algorithms (maximization), **we introduce an alternative, novel end-to-end synchronous data stream processing architecture consisting of tightly coupled sub-systems**. The resulting system, which we name *Blue Danube* works on the Application layer of the OSI model (Layer 7), interfacing with the rest of the layers through MPI, and establishes a new paradigm for efficient Large-Scale, Ultra-Low Latency data stream processing. Its theoretical model provides guarantees for latency bounds and zero loss. The synchronous Blue Danube system tightly integrates within its sub-systems the efficient real-time database Streamonas [12][13], following its semantic space-time models. In order to achieve end-to-end synchronization and real-time performance, a synchronous, high-bandwidth data stream channel is implemented with MPI. In order to strengthen the synchronous backbone communication of our architecture, we have performed our experimental analysis over the ARIS supercomputer of GRNET [29] which features an Infiniband interconnect.

The synchronous engine of Blue Danube, achieves real-time performance on the order of milliseconds, by applying cyclical synchronization techniques, over large-scale data streams on the order of Gbit/sec. The overall Blue Danube architecture is presented in section V.

B. *Large-Scale simulations – Intelligent Transportation Systems*

Large-Scale simulations have been performed in order to stress-test and benchmark the performance of Blue Danube. For these simulations we have used the simulation data and application requirements of the Linear Road Benchmark replicated multiple times in order to generate data streams on the order of Gbit/s. The Linear Road Benchmark (LRB) was developed by the researchers of works [21][22] in collaboration with the MIT Intelligent Transport Systems Program [46] using MITSIMLab [47] -- a simulation-based laboratory developed for evaluating the impacts of alternative traffic management system designs at the operational level and assisting in subsequent refinement. Over this very high bandwidth of streamed data, we ran the suite of algorithms specified in the Linear Road Benchmark, which includes accident avoidance. The scale of our experiments was very large, as in our simulations we process data streams generated by a large number of vehicles (478 million cars), almost half of the vehicles used globally at the time this experiment was concluded. The end-to-end latency achieved in our simulation results, presented in section VI, was on the order of milliseconds.

C. *Contributions*

- We have introduced a novel, End-to-End Synchronous, Distributed Data Stream Processing Architecture for Time-Sensitive Applications.
  The architecture enables Large-Scale (cluster of tens of nodes), High-Bandwidth (on the order of Gbit/s), Ultra-Low Latency (on the order of a few milliseconds) Data Stream Processing.
- We have introduced a novel, Cyclic Synchronization Data Stream Processing model, enabling the parallel, real-time processing of multiple high-bandwidth data streams,

semantically integrated into a single Large-Scale distributed database object. Based on the model, the main synchronous engine of the overall architecture has been developed.

- We have successfully performed a Large-Scale, Real-Time, Big Data Stream analysis experiment (4.7 Gbit/s total average aggregated throughput, 1 Terabyte in-memory distributed database, 4 milliseconds average query latency) demonstrating the suitability of our architecture for Time-Sensitive Applications such as accident avoidance for the Internet of Vehicles.

II. Related Work

Specific to online data processing architectures, the survey of [30] provides a structure of four generations of stream processing engines. Based on this survey, the First generation (initiated in the early 2000s) was characterized by extensions to traditional DBMSs, the Second generation (initiated in the mid-2000s) was characterized by distributed execution, the Third generation (initiated around 2010) allowed for the specification of User Defined Functions, while the Fourth generation is currently emerging (initiated around 2016), with architectures mixing elements deployed on edge computing resources. To this structure of generations published in [30] in 2017, we are adding a Fifth generation of architectures that have recently started to emerge, having as goal the effective ultra-low latency data stream processing of very high-bandwidth data streams.

The demand for ultra-low latency by present and future applications with limits from a few microseconds to a few milliseconds is highlighted in [1]. Five latency critical IoT use cases (factory automation, process automation, Smart Grids, Intelligent Transportation Systems and professional audio) and their respective requirements are analyzed in [4]. A survey on Internet of Vehicles, which is an emerging concept in Intelligent Transportation Systems, along with their safety applications categorization is provided in [8]. The survey highlights the importance of safety applications for saving lives and their susceptibility to delay.

Current stream processing solutions use a highly distributed environment on the cloud and through edge computing. For example, processing in Apache Storm [36] is represented as a directed acyclic graph with spouts ingesting data from various data sources, and bolts processing data. While multiple inputs can be forwarded to different nodes of the graph, Apache Storm does not use a real-time database like Blue Danube. Thus, the system cannot provide an integrated view of all accessible data from input streams – a problem inherent in systems using an operator-graph model.

Apache Flink [37] also performs stream processing through parallel dataflows. Dataflows are transformed by transformation operators with applications structured as Directed Acyclic Graphs. In a manner similar to Apache Storm, Apache Flink also does not use a real-time database. In Kafka [38], stream processing is performed through an application's processor topology. A topology is a graph of stream processors. Multiple inputs can be injected into different topologies while local data stores maintain state. Output streams from each topology can be forwarded to multiple output streams.

Benchmarking of Apache Storm, Apache Spark and Apache Flink has been performed in [9] using data from an online video game application. The results of the benchmarking, report event time latencies (end-to-end latencies) for all three systems are on the order of seconds.

As analyzed in [9], these latencies included various forms of delays:
(i) delays until windows were formed completely before being processed,
(ii) delays at the message broker and
(iii) delays due to queues at data generators.

Long end-to-end latencies between 60-90 seconds were also reported in [10] when benchmarking Microsoft Azure IoT Edge. These delays are due to the fact that Azure IoT service batches messages from an edge device in the IoT Hub on the cloud and writes the results from the multiple messages as a batch in a single blob file.

Being consistent with the history of temporal databases and stream processing, we should also include research on data streams before year 2000. More specifically, an extensive bibliography on spatio-temporal databases can be found in [39]. A consolidation approach to temporal data models and calculus-based query languages is provided in [40]. Research works for modeling time and time granularities in Databases, Data Mining and Temporal Reasoning, are presented in [41]. In still earlier related work, [42] introduced a stream processing paradigm of functional transformations (transducers) on streams. The authors in [43] describe the Tangram stream processor. Programmable transducers on streams are used throughout, providing a database flow computation capability that can be generalized.

The survey in [30] provides an additional classification of data stream processing engines in two categories: 'early stream processing solutions' and 'current stream processing solutions'. We'll follow the same classification when referring to related work and we'll also use the generic term 'processing engines', as the survey does, in order to refer to the different names of the systems, included over the years in the rich data streams literature (e.g. Data Stream Management Systems, Stream Data Processing systems etc.).

In Aurora [24], tuples flow through a loop-free, directed graph of processing operations and the work introduced a fundamental classification for QoS graph types as Delay-based, Drop-based and Value-based. The work tries to maximize QoS towards guaranteed accuracy by applying advanced techniques such as semantic load shedding. Medusa [32] is a distributed version of Aurora intended to be used by multiple enterprises that operate in different administrate domains. Borealis [33] is a distributed system that inherits core stream processing functionality from Aurora and distribution functionality from Medusa. CACQ [34] implements continuously adaptive, continuous queries. The Stream Processing Core [35], also a data-flow system, enables the execution of multiple stream processing applications simultaneously on a large cluster of machines. Each application is expressed in terms of a data-flow graph consisting of processing elements (PEs) that consume and produce streams of data.

The STREAM project [44] Data Stream Management System prototype supports a large class of declarative continuous queries over continuous and traditionally stored data sets. Our early work Streamonas Data Stream Management System [12][13] performed efficient data stream management proving the validity of the theoretical spatio-temporal models it defined, demonstrating also the performance characteristics of its ST-Cuboids database. The Stream Mill Miner Data Stream Management System for knowledge discovery [45], provides a data stream mining workbench that combines the ease of specifying high level mining tasks with the performance and QoS guarantees of a Data Stream Management System.

In this work, Blue Danube, presented in the following sections, through its novel multiple input End-to-End Cyclic Synchronization architectural model, reaches new levels of supported throughput and database size, while maintaining ultra-low latency on the order of a few milliseconds.

III. Comparison of Blue Danube with previous state-of-the-art works

The authors of the survey in [30] provide a model of an architecture for online data analysis, which generally consists of multi-tiered systems that comprise many loosely coupled components. In their overview they also refer to storage (including In-Memory storage) as a solution for long-term batch analysis and analysis of streaming data or data at rest. They refer also to the general notion that Data Stream Processing systems are commonly designed to handle and perform one-pass processing of unbounded streams of data.

The Blue Danube architecture, proposed in this work, differs from this general framework in a particularly significant way: Blue Danube consists of **tightly integrated, synchronous sub-systems, directly connected to the network ports** of multiple high-bandwidth data streams with an aggregate throughput of multiple Gbit/sec. Through this tight integration and synchronization, **Blue Danube achieves ultra-low latency** on the order of a few milliseconds, without intermediate buffering from brokers that can result in bandwidth reduction and increased latencies as analyzed in [9].

Also, Blue Danube does not use DAGs for processing data. Processing is not performed in one-pass only. Rather, streaming data is stored in a distributed database in memory (database of Spatio-Temporal cuboids [12][13] discussed in section IV). In this specialized database for data streams, data elements can be accessed and re-processed multiple times in parallel by multiple queries, while they still continue their flow within the database itself. Through the tight integration referred in the previous paragraph, the in-memory storage of Spatio-Temporal cuboids (ST-Cuboids) is used for real-time processing, and not for data at rest or for batch data (there is no constraint for cuboids to be used for data at rest or batch data processing, but the main goal is to enable ultra-low latency data stream processing).

The survey in [30] refers to essentially two models currently in use by stream processing engines:
(i) the operator-graph model where data is processed by being ingested into a Direct Acyclic Graph of operators, and (ii) the micro-batch model where data is grouped during short intervals.

Blue Danube architecture differs from both above approaches. It does not use a Directed Acyclic Graph of operators, and it does not use the micro-batch model. It rather uses a distributed database of ST-Cuboids in memory as presented in the previous paragraphs. Both current (real-time data) and historical data are indexed as temporal sequences and can be accessed by a large number of applications in real-time, in parallel with response times on the order of milliseconds. Thus, to the above two models (operator-graph model, micro-batch model) we can add, as a third model, the 'real-time database' model, where both historical and real-time data are efficiently indexed and stored in memory for retrieval in real time.

The work [48] presents Apache Flink's core mechanisms for managing persistent, large-scale pipelines with large application state in production. The paper refers to the global snapshotting problem for a distributed systems modelled as a set of processes connected via data channels, abstractly represented as a directed graph of nodes and edges. The snapshotting mechanism presented in the work acquires a global view of the system periodically or upon demand allowing for coarse grained rollback recovery in a asynchronous manner. Blue Danube provides consistency and state information at database level following the theoretical frameworks of real-time consistency published in [13], decoupling streaming from querying and thus allowing queries to access the database without the need to use directed graphs. The access of the database is performed by queries in parallel. Blue Danube follows a Cyclic Synchronization Data Stream Processing model (presented in section IV) connected to a real-time database of spatio-temporal cuboids.

Structured Streaming, a high-level streaming API in Apache Spark, is presented in [49]. Structured Streaming is based on automatically incrementalizing a static relational query instead of building a DAG of physical operators. Structured Streaming also supports end-to-end real-time applications that integrate streaming with batch and interactive analysis. The work presents high throughput results while using the Yahoo! Streaming Benchmark. Blue Danube does not use incrementalization to update results in response to new data. Our system firstly appends new data to the real-time database of spatio-temporal cuboids. Through the single integrated view of the stored data, queries are enabled to access the whole volume of data stored in the distributed database (in our experiments 1 Terabyte in-memory database). The whole cycle of database consistency and real-time processing is controlled by our novel Cyclic Synchronization Data Stream Processing model which enables optimization along the three dimensions of end-to-end latency, throughput and level of sophistication of the data analysis performed (Linear Road Benchmark suite of algorithms in our experiments).

The ultra-low latency performance characteristics of Blue Danube are attributed to the following factors:

- Blue Danube follows a novel Cyclic Synchronization Data Stream Processing model (section IV) which aims to minimize the latency of a single cycle of operations and guarantees that all necessary operations for database consistency based on latest information are performed within this cycle properly. The Cyclic Synchronization Data Stream Processing model is always monitored in order to maintain ultra-low latency conformance.

- In order to maintain the ultra-low latency characteristics of the Cyclic Synchronization Data Stream Processing engine, we have designed and implemented specialized sub-systems of POSIX threads and MPI calls and a locking mechanism which tightly integrates them (sections IV, V). We need to highlight that the locking operations, the processing by the POSIX threads and the MPI calls are performed within the same synchronization cycle, where we continuously monitor their ultra-low latency performance in an end-to-end manner.

IV. High-Throughghput Cyclic Synchronization Data Stream Processing

Blue Danube is implemented based on its novel Cyclic Synchronization Data Stream Processing model. The model aims to minimize the latency of a single cycle of operations and guarantees that all necessary operations for database consistency based on latest information are performed within the same synchronization cycle, where we continuously monitor their ultra-low latency performance in an end-to-end manner. In order to maintain the ultra-low latency characteristics we have designed and implemented specialized sub-systems of POSIX threads and MPI calls. These subsystems are tightly integrated with a locking mechanism. The locking operations, the processing by the POSIX threads and the MPI calls are all performed within the same synchronization cycle.

Our novel synchronization model of the Blue Danube architecture, based on which the respective engine has been implemented, is presented in Figure 1. Figure 1-(i) demonstrates the model in more detail, while Figure 1-(ii) provides the general symbol of the same model used in the overall architecture presented in Figure 2. As presented on the left side of Figure 1-(i) multiple cores running parallel threads at the Sender Node pre-process incoming streaming data from a high-bandwidth source.

Pre-processing involves the preparation of the data for storage in the distributed real-time database, hosted by the Receiver nodes. The distributed real-time database follows a fragmentation scheme and thus data are stored in different fragments. Through pre-processing these data are appropriately packaged for transmission through MPI based on this schema.

Synchronous MPI calls establish a very high bandwidth data stream channel between a Sender node (point A) and a Receiver node (point B). At points A and B a hybrid synchronization is performed. More precisely, at point A the parallel threads running at the cores of the node (POSIX threads) are synchronized with the process running the synchronous MPI_Send calls. For this purpose we have developed a specialized locking mechanism. The locking mechanism does not allow MPI_Send calls to deliver a data package, unless it is completely pre-processed in parallel by the multiple threads at the Sender node and becomes ready for delivery. Efficient

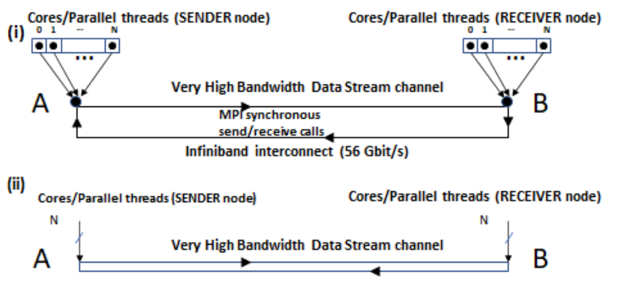

(i)
Cores/Parallel threads (SENDER node)
Cores/Parallel threads (RECEIVER node)
0 1 ... N
A
B
Very High Bandwidth Data Stream channel
MPI synchronous send/receive calls
Infiniband interconnect (56 Gbit/s)
(ii)
N
Very High Bandwidth Data Stream channel

Fig. 1. The novel End-to-End Cyclic Synchronization Data Stream Processing model.

pre-processing algorithms are of critical importance, as they affect the frequency of the Send-Receive cycles performed by the MPI calls.

At the Receiver node, similar hybrid synchronization is performed at point B. Once a data packet is received through MPI, our locking mechanism releases the multiple threads running at the cores of the Receiver node, which then process in parallel the data stored in the MPI package, storing them in their respective database fragment hosted by the same Receiver node. The MPI cycle waits until all parallel threads complete their tasks which include database consistency operations as also query processing.

Once all threads have completed their tasks over their associated database fragments, they release their locks in order for MPI to proceed to the next Send-Receive cycle. Earlier in this section, we have highlighted the criticality of the pre-processing algorithms in real-time performance at the Sender node.

The same criticality exists for the database consistency operations and query processing at the Receiver node: In case these operations (database consistency operations, query processing) are not efficient, they can create delays to the MPI cycle, resulting in higher latency -- missing the ultra-low latency goal, lowering processing frequency, and lowering throughput to be served.

In order to achieve efficient database consistency operations, we have integrated to the system a real-time database of spatio-temporal cuboids (ST-Cuboids) as shown in Figure 2. ST-Cuboids are indexed data structures stored in memory. They allow for random access and re-usability of any data element stored in them, enabling ultra-low latency characteristics as required by Time-Sensitive applications.

This real-time performance of ST-Cuboids has been extensively tested in distributed environments as published in the work [13]. ST-Cuboids greatly differ from data lakes, which are schema-agnostic big data repositories [15][16]. Data lakes can also consist of a massive collection of datasets that may vary in their formats [17]. On the contrary, ST-Cuboids follow a strict schema which allows efficient query processing and precise semantics. In order to achieve efficient query processing, we have used our well tested for real-time performance suite of algorithms which make use of the object oriented structure of the real-time database. Integration of these algorithms to the

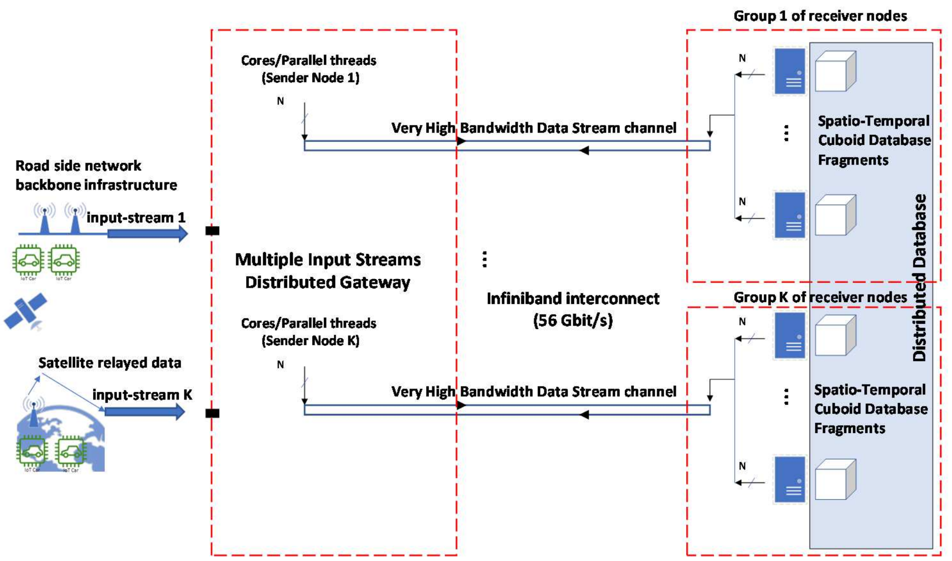

Group 1 of receiver nodes
Cores/Parallel threads (Sender Node 1)
N
Very High Bandwidth Data Stream channel
Spatio-Temporal Cuboid Database Fragments
Road side network backbone infrastructure
input-stream 1
Multiple Input Streams Distributed Gateway
Infiniband interconnect (56 Gbit/s)
Group K of receiver nodes
Distributed Database
Cores/Parallel threads (Sender Node K)
Satellite relayed data
input-stream K
Very High Bandwidth Data Stream channel
Spatio-Temporal Cuboid Database Fragments

Fig. 2. The End-to-End Synchronous, Distributed Data Stream Processing Architecture, Blue Danube, featuring multiple, parallel Cyclic Synchronization mechanisms, and a multiple input streams Distributed Gateway.

system was followed by extensive benchmarking tests which measured the overall performance of the system, verifying its suitability for Time-Sensitive applications (section VI).

Based on the model described in this section, a Cyclic Synchronization Data Stream Processing engine has been developed. Multiple instances of this engine run in parallel, performing large-scale, end-to-end, ultra-low latency data stream processing. The engine consists the backbone of the Distributed Architecture of Blue Danube and is extensively presented in the following section.

V. The End-to-End Synchronous Distributed Data Stream Processing Architecture

Our novel End-to-End Synchronous Distributed Architecture for Data Stream Processing is presented in Figure 2. The architecture uses multiple engines (instances of the engine analyzed in section IV) performing Cyclic Synchronization Data Stream Processing in parallel. The multiple parallel Cyclic Synchronization engines are used to scale-up deployments while maintaining the real-time characteristics of the model. In order to satisfy scalability needs, a Distributed Gateway (Figure 2) has been developed which receives multiple high-bandwidth input streams from the network infrastructures of Time-Sensitive applications.

In our experiments we have developed a 4-node Distributed Gateway with each one of the nodes being capable to process approximately 1 Gbit/s of streaming data. Each one of the nodes of the Distributed Gateway is associated with a number of database nodes (between 10 to 12 database nodes per Gateway node) forming a subgroup within the whole cluster (Figure 2). Communication between each one of the Gateway nodes and its associated group of database nodes is performed synchronously, through the Cyclic Synchronization Data Stream Processing engine, featuring a hybrid MPI/POSIX threads parallel environment as presented in the previous section.

In our experiments we have used 50 nodes, structured in 4 subgroups having 4 Gateway nodes and 46 database nodes (Figure 2). As already mentioned, each one of the subgroups is connected to its own independent Cyclic Synchronization engine. The 4 subgroups work in parallel without the need of any further synchronization among them, maintaining their ultra-low latency characteristics and allowing for higher scalability and bandwidth. The Distributed Gateway can scale-up linearly with only constraints the number of available nodes in the cluster and the Infiniband bandwidth. The Receiver nodes host the fragments of the real-time database. Despite the fact that these fragments are populated independently and in parallel for each subgroup of database nodes (through the engines), the database consists a single distributed database object, which integrates the whole spatio-temporal volume of the stored data. This means that despite the parallel subgroups processing information independently, a single integrated view of the stored data, is provided by the distributed database which maintains temporal semantics. The overall integrated architecture achieves ultra-low latency on the order of milliseconds.

Figure 2 shows the network infrastructure for an Internet of Things/Internet of Vehicles environment. The architecture supports multiple high-bandwidth inputs, directly connected to hubs of network infrastructures similar to [18][19]. As mentioned earlier, a unique characteristic of the architecture is that the multiple high-bandwidth input data streams are forwarded to a database of Spatio-Temporal cuboids, which provides a single integrated view (distributed in nature) of the stored data. During our Linear Road Benchmark experiments, by efficiently scaling up the available resources, the Distributed Gateway enabled the increase of the supported Internet of Things objects from 75 million smart objects (cars) which could be supported by a single gateway node, to the much higher level of 478 million smart objects (cars), while maintaining ultra-low latency on the order of milliseconds. In order to conceptually realize the level of the achieved scale we took into consideration that the number of motor vehicles registered in the US in year 2019 was approx. 276 million [20], a load that our experiments far exceeded. Cyclic synchronization can be identified when

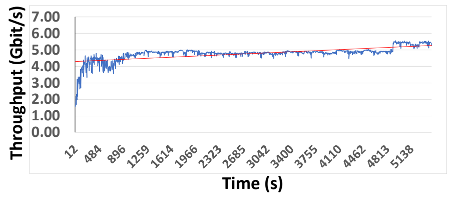


Fig. 3. Overall throughput processed by the 4 nodes of the Distributed Gateway.

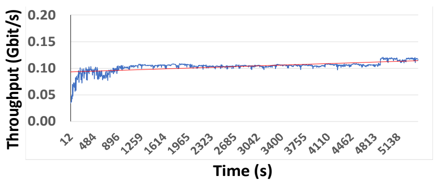
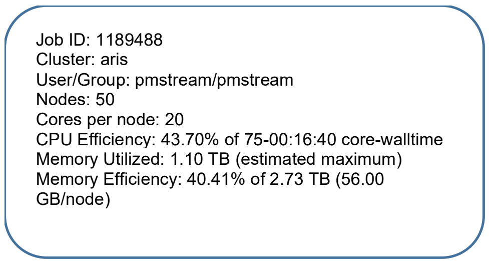


Fig. 4. Throughput processed by each one of the 46 Database nodes.



Job ID: 1189488
Cluster: aris
User/Group: pmstream/pmstream
Nodes: 50
Cores per node: 20
CPU Efficiency: 43.70% of 75-00:16:40 core-walltime
Memory Utilized: 1.10 TB (estimated maximum)
Memory Efficiency: 40.41% of 2.73 TB (56.00 GB/node)

Fig. 5. Extract of the output of the “seff” command.

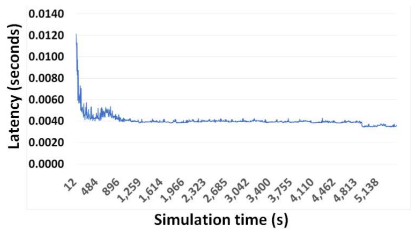


Fig. 6. Latency monitoring at the completion of each synchronization cycle.

comparing the throughput diagrams of the Distributed Gateway with each one of the throughput diagrams of the 46 database nodes (Figures 3 and 4). The two patterns are similar as fluctuations in the transmission of the MPI packages by the Distributed Gateway correspond to equivalent fluctuations of throughput at the database nodes. In all cases, the transmission throughput by the Distributed Gateway is equal to the summation of the individual receiving throughputs at each one of the database nodes.

Between each Distributed Gateway node (Sender) and the nodes in its associated group (Figure 2), the following Cyclic Synchronization Data Stream Processing operations (operations 1-5) take place:

1. Parallel pre-processing of a high-bandwidth incoming stream by multiple threads at a Distributed Gateway node preparing MPI packages for delivery to the respective Receiver nodes (database nodes) of its associated group;
2. Synchronization of threads at the Distributed Gateway node with the MPI process (point A of Figure 1) for sending the MPI packages to the respective Receiver nodes (database nodes) of its associated group;
3. On each one of the Receiver (database) nodes of a group, parallel database consistency operations and real-time query processing by threads running on each node;
4. Synchronization of threads at each one of the Receiver (database) nodes with the MPI process (point B of Figure 1) for continuing the Send-Receive MPI cycle;
5. Synchronization of a Distributed Gateway node (Sender) with the respective Receiving nodes (Database nodes) in its associated group through the MPI Send-Receive cycle.

The groups perform their respective synchronization cycles in parallel (Figure 2) which can be expressed as:

Parallel Cyclic Synchronization Data Stream Processing for each Group i

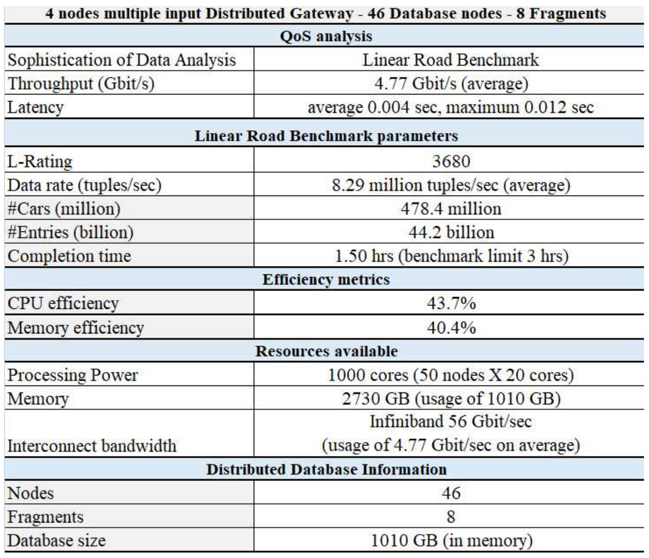
The architecture has been stress-tested with very-high bandwidth experiments demonstrating latency characteristics on the order of a few milliseconds, as presented in the following section.

VI. Very-High Bandwidth, Ultra-Low Latency Data Stream Analysis Experiments

In order to test the suitability of our novel End-to-End Synchronous Distributed Data Stream Processing Architecture for supporting Time-Sensitive applications, we have performed a series of experiments while running the Linear Road Benchmark (LRB) [21][22], simulating an Intelligent Transportation Systems Smart City infrastructure. Within the context of this benchmarking, and in order to simulate the direct connection to 4 high-bandwidth network ports, we used 4 copies of the input dataset of the LRB directly accessed by 4 nodes of the Distributed Gateway of Blue Danube. Each one of the 4 datasets includes simulation data for 10 expressways. In order to achieve the expected very-high-bandwidth required by the simulation, the input was regenerated and multiplied internally by the 4 nodes, by using a varying number of multiple ‘data generating’ threads (8-19 threads) bound to respective cores, accessing the input files in parallel. The aggregate data rate of the data stream exceeded 4 Gbit/s. A large number of early and current stream processing engines have used the LRB traffic simulation environment, which is challenging in terms of burstiness and real-time requirements. The benchmark was characterized by [23] as a milestone of stream benchmarks. LRB was introduced with experimental results for the Aurora system [24]. SCSQ-PLR [25] (in year 2011) was the first distributed system to have reached the LRB record level of 512 expressways (L=512). In the same publication a list of performance results from various systems is presented. Our early work Distributed Streamonas [13] reached a level of 550 expressways on LRB in a small-scale experiment of 5 nodes with a single input stream, limited by a 1 Gbit/s interconnect. Recently (in 2016) IBM ran the LRB [26][27] using Redis to maintain state and IBM Streams to handle the incoming events and queries. The performance of this architecture reached 400 thousand tuples/sec. Another recent Linear Road

Benchmark implementation was reported in [28] on Apache Flink (in 2019).

Our experiment, presented in this work, was performed on the ARIS supercomputer, deployed and operated by National Infrastructures for Research and Technology S.A. ARIS consists of 532 computational nodes, divided into four 'islands' [29]. For our experiments we have used 50 thin nodes, each having 20 cores with processor base frequency 2.8 GHz and 56 GB of RAM. ARIS has an Infiniband FDR14 network interconnect which reaches a speed of up to 56 Gbit/s making it ideal for implementing the very high-bandwidth data stream channel of the architecture (Figure 2). The topology we configured for the experiment has a 4 node Distributed Gateway, 46 database nodes, using 8 cores per node to serve 8 respective database fragments in an 1-1 association (4-46-8 topology). We used the data generated by the Linear Road Benchmark for the maximum level of expressways it supports, i.e. 10 expressways. Data generation (multiplication) by the multiple parallel threads is immediately followed by pre-processing from the Distributed Gateway, simulating the tight connection of the gateway to the network port. There are no provisions for queueing, thus minimizing latency. The average data rate received by the network ports, based on their specifications, is exceeded by the average processing rate of the Distributed Gateway. This differs from a streaming model where brokers are involved with unanticipated spikes in latency due to buffering or restructuring, as explained in [9]. The topology defined in our experiment supports 368 database fragments (46 database nodes $\times$ 8 fragments). Taking into consideration that each fragment can host 10 expressways, the final number of expressways supported (L-rating) is 3680 (368 fragments $\times$ 10). Figure 3 provides the graph of the total bandwidth of all 4 nodes of the Distributed Gateway throughout the duration of the experiment. As presented in Figure 3, the overall average bandwidth of the system when running the LRB and conforming to its requirements reached 4.77Gbit/s. In the following paragraphs we provide a performance analysis of the different subsystems of Blue Danube. Figure 4, provides throughput measurements on the database nodes. As analyzed earlier in this document the patterns of Figure 3 and Figure 4 are similar due the cyclic synchronization performed (section IV). The average throughput of each one of the 46 database nodes is 0.10 Gbit/s. The sum of the individual throughputs at the 46 database nodes is equal to the overall throughput transmitted by the 4 gateway nodes. More specifically, based on isolated experiments over the very-high bandwidth data stream channel subsystem with MPI calls, we were able to exceed 50 Gbit/s of throughput. The throughput was reduced from that maximum value to the value of 4.7 Gbit/s due to: (i) the overhead incurred by the locking system, (ii) the overhead for pre-processing data by the Distributed Gateway nodes for transmission through MPI and (iii) the sophistication of the algorithms of the Linear Road Benchmark running on the Receiver nodes (database nodes). All above overheads contribute to the end-to-end latency measured within each cycle of MPI send and receive calls. The Distributed Data Gateway can pre-process and transmit on average 8.2 million tuples per second, with each one of the database nodes processing on average 0.18 million tuples/sec. In Figure 7 we present a tabulation of performance results which includes an analysis for the topology of the architecture -- which consists of a 4 node Distributed Gateway, 46 Database nodes and 8 Database Fragments (4-46-8 topology). As shown in the tabulation, the average latency is 4 milliseconds and the maximum latency is 12 milliseconds, demonstrating the ability of Blue Danube to handle time-critical events especially for IoV safety applications. Such time-critical events are included in the Linear Road Benchmark which requires the notification of car drivers moving towards a location where a car accident has occurred. Latency is measured from the moment data is generated until the whole synchronization cycle between two MPI send-receive calls completes (Figure 1, Figure 2). Queueing can create long latencies as researchers of [9] have proven. For this reason, the synchronous architecture internally does not use queues, as it processes one-by-one batches of data received by the system in a synchronous manner. The end-to-end latency measurements include any delays from internal buffering that may occur within the ARIS subsystems. In terms of Linear Road Benchmark-specific parameters, the L-Rating achieved in running the benchmark is 3680, i.e. the system can host effectively cars from 3680 XWays. As explained in previous sections, and repeated here with all related parameters listed in Figure 7, the distributed database hosts cars in fragments of 10 expressways. Each of the 46 database nodes serves 8 fragments. The total number of cars managed by the database is 478 million, a number which far exceeds the registered motor vehicles in the US in 2019 [20]. In order to provide real-time responsiveness we have designed and parameterized the system in a way that keeps the average processing rate higher than the specified average data generation rate. Based on this, the experiment consumes all data in approximately 1.50 hours, much earlier than the 3-hour duration expected by the benchmark (Figure 6). The database of Blue Danube is a distributed database of Spatio-Temporal cuboids capturing 1.01TB of memory (this number excludes the memory used by the Distributed Gateway nodes). The distributed database consists a single distributed database object which we can access, visualize and analyze in parallel (our real-time visualization results of the contents of the distributed database are being prepared for a future publication). In order to distribute



4 nodes multiple input Distributed Gateway - 46 Database nodes - 8 Fragments	
QoS analysis	
Sophistication of Data Analysis	Linear Road Benchmark
Throughput (Gbit/s)	4.77 Gbit/s (average)
Latency	average 0.004 sec, maximum 0.012 sec
Linear Road Benchmark parameters	
L-Rating	3680
Data rate (tuples/sec)	8.29 million tuples/sec (average)
#Cars (million)	478.4 million
#Entries (billion)	44.2 billion
Completion time	1.50 hrs (benchmark limit 3 hrs)
Efficiency metrics	
CPU efficiency	43.7%
Memory efficiency	40.4%
Resources available	
Processing Power	1000 cores (50 nodes X 20 cores)
Memory	2730 GB (usage of 1010 GB)
Interconnect bandwidth	Infiniband 56 Gbit/sec (usage of 4.77 Gbit/sec on average)
Distributed Database Information	
Nodes	46
Fragments	8
Database size	1010 GB (in memory)

Fig. 7. QoS analysis for the Blue Danube 4 - 46 – 8 topology.

workloads across the supercomputer, ARIS uses the SLURM workload manager (Simple Linux Utility for Resource Management) [14]. We have used the "seff" command of SLURM in order to receive information for the completed job which ran the experiment (Figure 5). In the utilization report we have received useful results for the usage of the resources and the efficiency of the system as follows: (i) The distributed database object of Spatio-Temporal cuboids hosting 478 million smart objects (cars) captured 1.01 TB of RAM (46 out of the 50 available nodes), (ii) by using 9 of the 20 available cores at the Distributed Gateway process -- and also 9 of the 20 available cores at each one of the 46 the database nodes -- the CPU efficiency was 43.7% and (iii) as the available memory in all 50 nodes was 2.73 TB the memory efficiency was 40.4%. Figure 6 presents the latency measured throughout the duration of the experiment. Maximum latency was recorded at the beginning of the simulation with an initial value of 12 milliseconds and as the cluster stabilizes, latency drops to lower levels. The average latency was 4 milliseconds. As mentioned earlier, latency is measured by the time an event is received by the system until the event completes the end-to-end synchronization cycle. The measurement was performed by the same Distributed Gateway node using the internal clock of the machine. As MPI messaging is used for implementing the very high-bandwidth data stream channel (Figure 2), within each cycle of average 4 milliseconds, 33120 tuples are processed. These tuples are included in 46 MPI messages pre-processed and sent to the nodes, each one containing 720 tuples structured in 8 fragments per database node. In practice latency is calculated every 1000 synchronization cycles by taking the respective average (time needed to complete 1000 synchronization cycles divided by 1000). Our experiments, while successful, have not used the maximum level of available resources. This is also reflected in the efficiency metrics presented in Figure 7. More specifically memory efficiency was 40.4%, CPU efficiency 43.7% and throughput efficiency 8.5% as the topology used 4.77Gbit/s of the available 56 Gbit/s. In our future work we plan to perform experiments that utilize available resources at higher efficiency levels and measure the impact this increase may have on latency. The 4.77 Gbit/s aggregated throughput achieved by Blue Danube is not an upper bound. With additional resources available (nodes) we could have scaled the aggregated throughput up to the Infiniband bandwidth of 56 Gbit/s. This could be performed by extending the distributed database and the Distributed Gateway with additional groups of 1 Gateway node per 10 database nodes. The results presented in Figure 7 are very successful, proving that our novel End-to-End Synchronous, Distributed Data Stream Processing Architecture can effectively serve at large scale, the needs of Time-Sensitive applications at the Application layer, guaranteeing packet transport with bounded latency, low packet delay variation, and zero packet loss.

VII. Conclusion

There is increasing demand for ultra-low latency by time-sensitive applications (including factory automation, process automation, Smart Grids, Intelligent Transportation Systems and professional audio) with limits from a few microseconds to a few milliseconds. Despite the currently evolving IEEE Time-Sensitive Networking standard, ultra-low latency results are not satisfactory when crossing the layers of the OSI model in an end-to-end manner. The problem becomes worse when crossing additional layers on the cloud. This latency gap is significant as recent surveys report latencies when benchmarking data stream processing solutions on the order of seconds or even, in some IoT platforms, between 60-70 seconds. Most of these solutions use loosely coupled components and asynchronous communication. Our work bridges the gap by introducing an alternative, novel **end-to-end synchronous data stream processing architecture** consisting of tightly coupled sub-systems. The resulting system, which we name Blue Danube, works on the Application layer of the OSI model (Layer 7) -- interfacing with the rest of the layers through MPI -- and establishes a new paradigm for efficient Large-Scale, Ultra-Low Latency data stream processing. Our system introduces a novel Cyclic Synchronization Data Stream Processing model, enabling the parallel, real-time processing of multiple high-bandwidth data streams, semantically integrated into a single large-scale distributed database object. We have successfully tested the system by performing a Large-Scale, Real-Time, Big Data Stream analysis experiment (4.7 Gbit/s total average aggregated throughput, 1 Terabyte in-memory distributed database, 4 milliseconds average query latency) demonstrating that our proposed architecture is suitable for Time-Sensitive Applications such as accident avoidance for the Internet of Vehicles.

Acknowledgment

The research leading to the results presented in this publication has received funding from the European Union's funded Project PolicyCLOUD under grant agreement No. 870675.

This work was supported by computational time granted from the National Infrastructures for Research and Technology S.A. (GRNET S.A.) in the National HPC facility - ARIS - under project IDs pr002042-Streamopolis and pr006056_thin-Streamopolis II.

References

[1] A. Nasrallah et al., "Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research," in *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 88-145, First quarter 2019.

[2] S. Tschöke, F. Lynker, H. Buhr, F. Schreiner, A. Willner, A. Vick, and M. Chemnitz, "Time-sensitive networking over metropolitan area networks for remote industrial control," in *Proc. of the 2021 IEEE/ACM 25th International Symposium on Distributed Simulation and Real Time Applications*, September 2021, 22: 1-4.

[3] 2022. Time-Sensitive Networking (TSN) Task Group. Time-Sensitive Networking (TSN) Task Group | (ieee802.org) .

[4] P. Schulz et al., "Latency Critical IoT Applications in 5G: Perspective on the Design of Radio Interface and Network Architecture," in *IEEE Communications Magazine*, vol. 55, no. 2, pp. 70-78, February 2017.

[5] V. Gowtham, O. Keil, A. Yeole, F. Schreiner, S. Tschöke, and A. Willner, "Determining edge node real-time capabilities," in *Proc. of the 2021 IEEE/ACM 25th International Symposium on Distributed Simulation and Real Time Applications*, September 2021, 21: 1-9.

[6] 2017. "Time-Sensitive Networking: A technical Introduction," *White Paper*, Cisco, https://www.cisco.com/c/dam/en/us/solutions/collateral/industry-solutions/white-paper-c11-738950.pdf .

[7] P. Johnson, "An OSI Model for Cloud", *Cisco Blogs*, February 2017, https://blogs.cisco.com/cloud/an-osi-model-for-cloud .

[8] S. Sharma, and B. Kaushik, "A survey on internet of vehicles: Applications, security issues & solutions," *Vehicular Communications, Elsevier*, Volume 20, 2019.

[9] J. Karimov, T. Rabl, A. Katsifodimos, R. Samarev, H. Heiskanen, and V. Markl, "Benchmarking Distributed Stream Data Processing Systems," in *Proc. of the IEEE 34th International Conference on Data Engineering (ICDE)*, 2018, 1507-1518.

[10] A. Das, S. Patterson, and M. Wittie, " EdgeBench: Benchmarking Edge Computing Platforms," *in Proc. of the IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, 2018, 175-180.

[11] J. Boner et al. "The Reactive Manifesto," 2014, www.reactivemanifesto.org .

[12] P.A. Michael, and D. S. Parker, "The Semantic Space-time models of the Streamonas Data Stream Management System," *in Proc. of the 2009 World Congress on Computer Science and Information Engineering (CSIE)*, 2009, IEEE, Los Angeles / Anaheim, USA.

[13] P.A. Michael, "Architecture, Data Model and Real-Time Performance Evaluation of the Streamonas Data Stream Management System,", *UCLA*, 2015, ProQuest ID: Michael_ucla_0031D_13221. Retrieved from escholarship.org/uc/item/32j8x58z .

[14] 2021. Running Jobs - ARIS DOCUMENTATION (grnet.gr). https://doc.aris.grnet.gr/run/ .

[15] Y. Zhang, and Z. G. Ives, "Finding Related Tables in Data Lakes for Interactive Data Science," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, 1951-1966.

[16] R. Hai, S. Geisler, and C. Quix, "Constance: An Intelligent Data Lake System," in *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*, 2016, 2097-2100.

[17] F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena, "Data lake management: challenges and opportunities," *in Proceedings of the VLDB Endownment*, 2019, Vol. 12, Issue 12, 1986-1989.

[18] 2021. XR-Optics-SN-0219-RevE-1220.pdf (infinera.com). https://www.infinera.com/wp-content/uploads/XR-Optics-SN-0219-RevE-1220.pdf .

[19] 2021. Enabling 400G everywhere: comparing IP-optical network use cases | Nokia. https://www.nokia.com/blog/enabling-400g-everywhere-comparing-ip-optical-network-use-cases/ .

[20] 2021. Automobile registrations - United 2019 | Statista . https://www.statista.com/statistics/192998/registered-passenger-cars-in-the-united-states-since-1975/ .

[21] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbets, "Linear Road: A Stream Data Management Benchmark," in *Proceedings of the 30th Intl. Conference on Very Large Data Bases*, 2004, Volume 30.

[22] 2021. Linear Road - Home. https://www.cs.brandeis.edu/~linearroad/ (2004).

[23] A. Pagliari, F. Huet, and G. Urvoy-Keller, "Towards a High-Level Description for Generating Stream Processing Benchmark Applications," in *Proceedings of the IEEE International Conference on Big Data (Big Data)*, 2019, 3711-3716.

[24] D.J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," in *The VLDB Journal,* 2003.

[25] E. Zeitler, and T. Risch, "Massive scale-out of expensive continuous queries," in *Proc. VLDB Endow*, 2011, 4: 1181-1188.

[26] 2021. Choosing the right platform for high-performance, cost-effective stream processing applications | IBM. https://www.ibm.com/cloud/blog/choosing-right-platform-high-performance-cost-effective-stream-processing-applications .

[27] 2021. Walmart & IBM Revisit the Linear Road Benchmark- Roger Rea, IBM (slideshare.net). https://www.slideshare.net/RedisLabs/walmart-ibm-revisit-the-linear-road-benchmark (2016).

[28] M. Hanif, H. Yoon, and C. Lee, "Benchmarking Tool for Modern Distributed Stream Processing Engines," in *Proceedings of the International Conference on Information Networking (ICOIN)*, 2019, 393-395.

[29] 2021. HPC | National HPC Infrastructure (grnet.gr). https://hpc.grnet.gr/en/ .

[30] M.D. de Assunção, A. da Silva Veith, and R. Buyya, "Distributed data stream processing and edge computing: A survey on resource elasticity and future directions," in *Journal of Network and Computer Applications*, 2018, Volume 103 (2018), 1-17.

[31] D.J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management", *in The VLDB Journal,* 2003.

[32] M. Balazinska, H. Balakrishnan, and M.Stonebraker, "Load Management and High Availability in the Medusa Distributed Stream Processing System", *in Proc. of the 2004 ACM SIGMOD Intl. Conf. on Management of Data*, 2004.

[33] D J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The design of the Borealis stream processing engine", in *Proc. of the 2nd Biennial Conference on Innovative Data Systems Research (CIDR),* Asilomar, CA, 2005.

[34] S. R. Madden, M. A. Shah, J. M. Hellerstein, and V. Raman. 2003. Continuously adaptive continuous queries over streams. In Proc. of the 2002 ACM SIGMOD Intl. Conference on Management of Data.

[35] N. Jain, L. Amini, H. Andrade, R. King, Y. park, P. Selo, and C. Venkatramani. 2006. Design, Implementation, and Evaluation of the Linear Road Benchmark on the Stream Processing Core. In Proc. of the 2006 ACM SIGMOD Intl. Conference on Management of Data.

[36] 2021. Apache Storm. https://storm.apache.org/ .

[37] 2021. Apache Flink: Stateful Computations over Data Streams. https://flink.apache.org/ .

[38] 2021. Apache Kafka. https://kafka.apache.org/ .

[39] C. Zaniolo, S.Ceri, C.Faloutsos, R.T. Snodgrass, V. S. Subrahmanian, and R. Zicari, "Advanced Database Systems", 1997, Morgan Kaufmann.

[40] R. T. Snodgrass (ed.), I. Ahn, G. Ariav, D.S. Batory, J. Clifford, C. E. Dyreson, R. Elmasri, F. Grandi, C. S. Jensen, W. Käfer, N. Kline, K. Kulkanri, C. Y. T. Leung, N. Lorentzos, J. F. Roddick, A. Segev, M. D. Soo, and S. M. Sripada. 1995. The TSQL2 Temporal Query Language. Kluwer Academic, Norwell, MA.

[41] C. Bettini, S. Jajodia, and S. X. Wang. 2000. Time Granularities in Databases, Data Mining, and Temporal Reasoning. Springer.

[42] D. S. Parker. Stream Data Analysis in Prolog. 1990. In L. Sterling, ed., The Practice of Prolog, Cambridge, MA: MIT Press.

[43] D. S. Parker, R. R. Muntz, and H. L. Chau. 1989. The Tangram stream query processing system. In Proc. of the Fifth International Conference on Data Engineering.

[44] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom. 2003. STREAM: The Stanford Stream Data Manager. In Proc. of the 2003 ACM SIGMOD Intl. Conf. on Management of Data.

[45] H. Thakkar, B. Mozafari, and C. Zaniolo, "Designing an Inductive Data Stream Management System: the Stream Mill Experience," *in Proc. of the Second International Workshop on Scalable Stream Processing Systems, Nantes, France,* 2008.

[46] 2022. Home | ITS (mit.edu). https://www.its.mit.edu/ .

[47] 2022. MITSIMLab : A Simulation-based Lab for Evaluating Impacts of Alternative Traffic Management System Designs | MIT Technology Licensing Office. https://tlo.mit.edu/technologies/mitsimlab-simulation-based-lab-evaluating-impacts-alternative-traffic-management-system .

[48] P. Carbone, S. Ewen, G. Fóra, S. Haridi, S. Richter, and K. Tzoumas, "State management in Apache Flink: Consistent stateful distributed stream processing," in PVLDB, 2017.

[49] M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, A. Ghodsi, I. Stoica, and M. Zaharia, "Structured Streaming: A declarative API for Real-Time applications in Apache Spark," in Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data, 2018, 601-613.