Electronics and Information Technologies, 2026, 35, 77–96
http://publications.lnu.edu.ua/collections/index.php/electronics/index

UDC: 004.75, 004.41

# A REVIEW OF CLOUD-BASED AUDIO PROCESSING USING DIRECTED ACYCLIC SIGNAL-PROCESSING GRAPHS IN MICROSERVICE ARCHITECTURES

***Oleh Chaplia* & *Halyna Klym****
*Lviv Polytechnic National University,*
*12 Stepan Bandera Street, Lviv, 79013, Ukraine*

Chaplia, O., & Klym, H. (2026). A Review of Cloud-Based Audio Processing Using Directed Acyclic Signal-Processing Graphs in Microservice Architectures. *Electronics and Information Technologies*, 35, 77–96. https://doi.org/10.30970/eli.35.6

## ABSTRACT

**Background**. Audio processing chains in digital audio workstations (DAWs) are directed acyclic graphs (DAGs) of digital signal processing (DSP) nodes. Executing such graphs in the cloud promises elastic compute, collaboration, and access from thin clients. Microservices dominate cloud systems, but their principles were shaped by request-and-response workloads, not by block-synchronous audio. This paper reviews approaches that process and render audio in the cloud as DSP DAGs and evaluates them against microservice principles.

**Materials and Methods**. A systematic protocol was applied to two complete proceedings volumes of the IEEE International Symposium on the Internet of Sounds, using database searches and backward snowballing. From 97 proceedings papers and 49 full texts from other sources, 43 peer-reviewed studies published from 2022 to 2026 were included and grouped into eight method families, and each method was assessed on execution location, service granularity, real-time mode, reported latency and its measurement point, scaling, fault tolerance, and support for DAW-originated graphs.

**Results and Discussion**. Browser DAWs keep the DAG on the client and use the cloud for storage and collaboration. Networked music performance systems relay streams within strict latency budgets but do not render offline graphs remotely. Edge rendering services move a single stage to the network edge. Audio microservices expose offline tasks through REST APIs. Cloud-native media pipelines run containerized graphs with self-healing, and generic DAG engines provide scheduling and scaling but ignore sample-accurate timing. None of the reviewed systems execute a DAW-originated DSP DAG as microservices with block-level deadlines. Reported latencies are measured at different points and are not comparable. Cloud and container systems report hundreds of milliseconds to seconds.

**Conclusion**. Existing systems lack a portable representation of DAW graphs, a granularity model for stateful DSP nodes, deadline-aware scheduling in container orchestrators, low-latency block transport between services, and benchmarks for cloud audio rendering. These gaps define the research agenda for microservice-based cloud audio processing.

***Keywords***: cloud computing, microservices, digital signal processing, directed acyclic graph, digital audio workstation, reliability

## INTRODUCTION

Audio production, broadcasting, game audio, virtual reality, and music education rely on chains of digital signal processing (DSP) operations, and in a digital audio workstation

© 2026 Oleh Chaplia & Halyna Klym. Published by the Ivan Franko National University of Lviv on behalf of Електроніка та інформаційні технології / Electronics and Information Technologies. This is an Open Access article distributed under the terms of the Creative Commons Attribution 4.0 License which permits unrestricted reuse, distribution, and reproduction in any medium, provided the original work is properly cited.

(DAW) such a chain is a graph. Tracks feed insert effects, sends feed auxiliary buses, buses feed the master chain, and the master chain feeds a render or a playback device. Because feedback loops are not allowed inside the processing chain, the graph is a directed acyclic graph (DAG). The DAW engine sorts the graph topologically and executes it block by block. The same computation model is used by web audio engines, modular synthesis environments and media frameworks [1, 2]. Immersive mixes for object-based and binaural delivery add object renderers and bed buses to the same graph, and practitioners already manage such sessions in their DAWs [3].

Cloud computing has changed how such systems are built and delivered. The Internet of Sounds vision describes networks of sound devices that sense, process, and exchange audio with the help of edge and cloud resources [1]. Remote and cloud-based production has moved from proposals to deployed systems, as the review of remote immersive audio production and the cloud-based delivery systems in the broadcast literature documents [4, 5]. Remote immersive audio production extends this trend to many channels transported to a remote mixing site and to shared virtual spaces with many concurrent listeners [4]. Browser-based DAWs have reached a level where multitrack recording, plugins, and offline rendering work without a native installation [2]. At the same time, the dominant architectural style for cloud software is the microservice architecture. Microservices offer independent deployment, horizontal scaling, technology heterogeneity, and fault isolation [6, 7]. These qualities are attractive for a cloud audio engine that must serve many users and many sessions at the same time.

However, audio DSP is a poor fit for the assumptions behind classic microservices. A block of 128 samples at 48 kHz lasts 2.67 ms. Musicians who play together over a network tolerate a mouth-to-ear delay of 20 to 30 ms [8]. A cloud microservice call, a container restart, or a serverless cold start takes tens to thousands of milliseconds. This is the tension that any cloud audio engine must resolve. DSP nodes such as filters, delays, and reverbs hold internal state that must follow the stream. Every remote edge in the graph adds a network hop. Retries, circuit breakers, and autoscaling reactions operate at time scales that are much longer than a block period. A cloud audio system therefore needs a careful mapping between the DSP DAG and the set of services that execute it.

Several research directions touch this problem from different sides. GPU-oriented DAW APIs execute audio graphs on accelerators [9]. Web Audio Modules bring a plugin standard to browser DAWs [2]. Microservice backends were used to add generative music functions to a DAW through a VST plugin [10], and DSP as a service was proposed as a general model for server-based music production [11]. Edge computing was applied to spatial audio rendering [12]. Cloud native media pipelines were containerized and orchestrated in Kubernetes [13]. Generic DAG engines for serverless and stream processing workloads mature quickly [14, 15]. Two recent works cover neighboring ground. Turchet et al. survey the Internet of Sounds as a whole, including its edge and cloud layers [1]. Turchet and Krstulovic propose the vision of DSP as a service, which moves music production tasks to servers, segments such services into asynchronous, reactive, and low-latency tiers, and analyzes three prototypes under latency and reliability [11]. Neither takes the microservice architecture or the DAW origin of the graph as its object. Still, no review brings these works together and asks how a DSP DAG that originates from a DAW can be executed as a set of cloud microservices.

Remote and cloud production workflows are now documented in peer-reviewed work rather than only in vendor material [4, 5]. Browser DAWs reached multitrack recording, plugin hosting, and offline rendering between 2023 and 2026 [16, 2, 17]. Container orchestration and serverless platforms matured to the point where containerized media graphs run in production with self-healing [13]. The building blocks exist, but the mapping between them and a DSP DAG has not been studied.

This paper fills this gap with three contributions.

1. A taxonomy of eight solution families for cloud, edge, and browser-based execution of audio DSP graphs, derived from 43 peer-reviewed studies selected with an explicit protocol.
2. A mapping of eight microservice principles to the requirements of DSP DAG execution, which classifies each principle as conflicting, neutral, or requiring an audio-specific interpretation.
3. A research agenda of eight gaps, ranked by the number of included studies that touch each gap.

The goal is to review existing solutions for processing and rendering audio in the cloud as DSP DAG graphs, to describe their models, methodologies, advantages and drawbacks, to compare them, and to identify what such systems lack today. The focus is on the microservice architecture and its principles applied to audio DSP graphs, including graphs that originate from DAW sessions. The rest of the paper is organized as follows. The Materials and Methods section describes the review method. The Results and Discussion section presents the solution families, the comparison, the mapping of microservice principles to DSP DAG requirements, the answers to the research questions, the open problems, and the threats to validity. The Conclusion summarizes what is missing and outlines further research.

## MATERIALS AND METHODS

### Review protocol

This study is a systematic literature review with a narrative synthesis. The protocol defines the sources, the search strings, the inclusion and exclusion criteria, the quality checklist, and the data extraction fields before the search. Three search stages were used. Stage 1 screened two complete proceedings volumes, the 2023 4th International Symposium on the Internet of Sounds and the 2024 IEEE 5th International Symposium on the Internet of Sounds, because this venue concentrates the peer-reviewed work on networked, edge, and web-based audio. Stage 2 searched the publisher sites of IEEE Xplore, ScienceDirect, Springer Link, ACM Digital Library, MDPI, the Journal of the Audio Engineering Society, and the SMPTE Motion Imaging Journal using a general web search engine, and checked Scopus indexing for every candidate on the publisher page. The search strings combined one term from the first group with one term from the second group. The first group was "cloud", "microservice", "serverless", "edge computing", "container", "Kubernetes" and "DAG". The second group was "audio", "DSP", "audio graph", "digital audio workstation", "networked music performance", "spatial audio rendering", "media pipeline" and "stream processing". Stage 3 applied backward snowballing to the reference lists of the included full texts. The search was performed in September 2026.

The inclusion and exclusion criteria are the following.

1. Included. Journal articles indexed in Scopus, or papers in IEEE or ACM conference proceedings, published from 2023 to 2026.
2. Included. The paper describes, implements, or evaluates a system, architecture, protocol, or model for processing, rendering, or transporting audio, or for executing processing graphs, in a cloud, edge, container, browser, or networked setting.
3. Excluded. Preprints, vendor documentation, white papers, magazine articles, and product announcements.
4. Excluded. Conference papers not published by IEEE or ACM.
5. Excluded. Papers on music information retrieval, perception, or datasets without a system architecture component.
6. Excluded. Papers whose full text was not accessible to the authors, so that the extraction fields could not be verified.

One exception to the time window was accepted. The journal article by Bianchi et al. from 2022 was included through snowballing because it is the only work that formalizes the DAW graph for parallel execution [9]. Each included paper was checked against a quality

checklist with four items. The venue is peer-reviewed. The system or model is described in enough detail to classify it. An evaluation or a measurement is reported, or the paper is a position or review paper that is marked as such. The reported latency, if any, has a stated measurement point. Papers that failed the second item were excluded. The other items were recorded and are used in the comparison.

For every included paper, the following fields were extracted into a table. Solution family, DAG location, whether the DAG is explicit, service granularity, real-time mode, reported latency with its measurement point, scaling mechanism, fault tolerance mechanism, DAW origin, and the research gaps the paper touches. The screening and the extraction were performed by one reviewer. The family assignment was done by open coding of the extracted fields followed by grouping into families, and the grouping was revised twice until every paper fitted one family. Then an extraction table with all fields and the quality checklist outcome per study, and the study-to-gap matrix that produces the gap ranking were created.

## Selection results

**Fig. 1** shows the selection flow. Stage 1 screened 97 proceedings papers by title and abstract, 47 from the 2023 volume and 50 from the 2024 volume. 36 papers met the topic criteria. 19 of them were excluded because their full text was not accessible, so 17 were included, and 80 were excluded in total. 61 were excluded for music information retrieval, perception studies, and datasets without a system component, and 19 for missing full-text access. Stage 2 ran 31 query executions that returned about 300 result entries, which were screened by title and snippet, and 49 records were assessed in full text. 21 records were included, and 28 were excluded: 9 as preprints, 5 outside the time window, 3 as workshop or magazine articles, 3 as vendor material, and 8 for missing full-text access. Stage 3 added 5 records by backward snowballing after 2 candidates were excluded for missing full-text access. In total, 43 peer-reviewed studies were included. 17 come from the two proceedings volumes and 26 from journals and other IEEE or ACM proceedings. All 43 passed the first two checklist items. 31 report an evaluation or a measurement, and 12 are

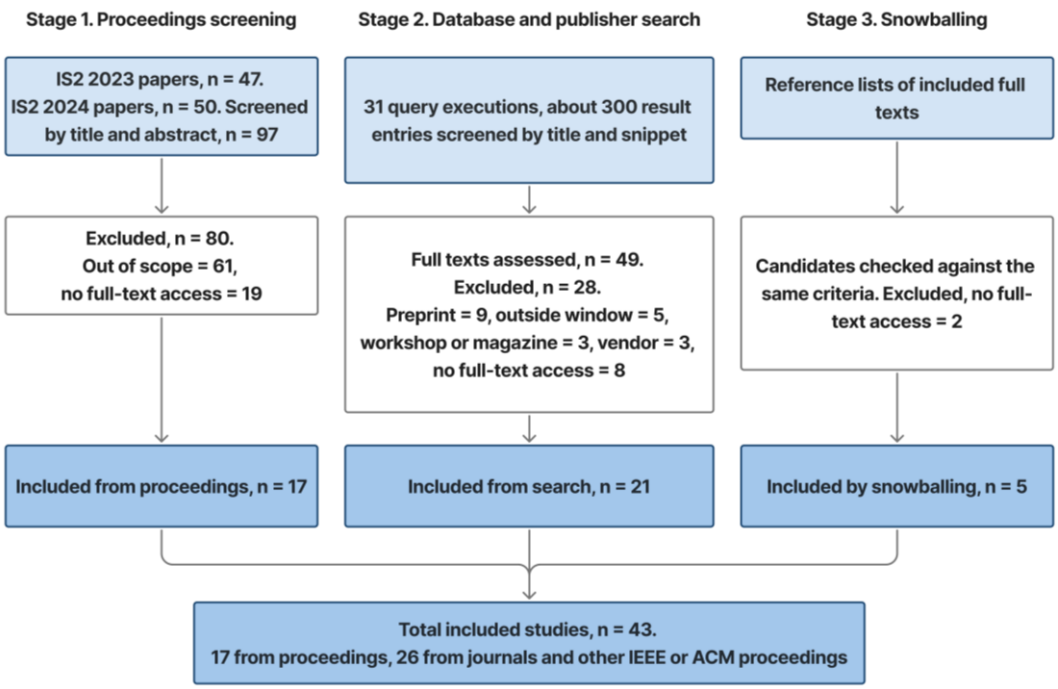


**Fig. 1.** Selection flow of the review with the number of records at each stage

position, review, or design papers. 24 reports a latency together with its measurement point. **Fig. 2** shows the distribution of the included studies over the eight solution families and the three sources. The hatched segments show that the audio microservice, media pipeline, and generic DAG engine families come entirely from outside the symposium, whereas the fog family comes entirely from it.

## Research questions

The review answers the following research questions.

1. RQ1. Which existing solutions process or render audio in the cloud, at the edge, or in the browser as a DSP DAG, and which computation model do they use?
2. RQ2. Which of the microservice principles conflict with block-synchronous execution of a DSP DAG, for which principles does the literature contain a workaround, and which principles apply without change?
3. RQ3. Which latencies do the solution families report, at which point were they measured, and are they comparable across families?
4. RQ4. What is missing for executing a DAW-originated DSP DAG as cloud microservices, and how many included studies touch each missing element?

## Comparison criteria

The comparison criteria are the following.

- DAG location. Client, edge, cloud, or a combination.
- Explicit DAG. Whether the solution represents the processing chain as an explicit graph that can be inspected, serialized, and scheduled.
- Service granularity. Monolith, one service per task, one service per DSP node type, or one service per subgraph.
- Real-time mode. Block-synchronous processing, streaming with a latency budget, or offline rendering.
- Reported latency, its measurement point, and the measurement method.
- Scaling and fault tolerance mechanisms.
- Support for DAGs that originate from a DAW session or a DAW plugin format.

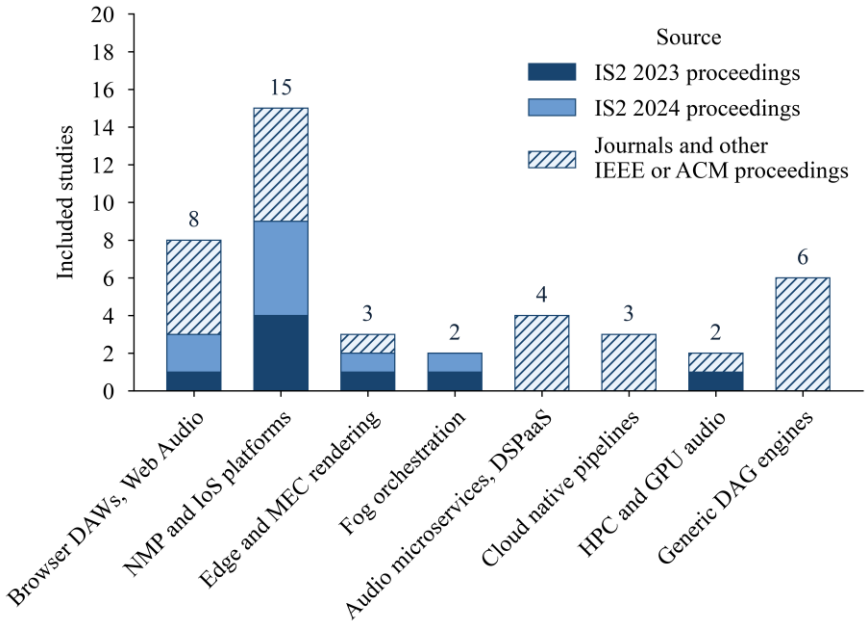


**Fig. 2.** Distribution of the 43 included studies over the solution families and the three sources of the review

## RESULTS AND DISCUSSION

The eight solution families are presented from the client to the cluster. Browser DAWs execute the graph on the client. Networked music performance systems keep the graph on the client and move only streams. Edge and fog systems move one stage to the network edge. Audio microservices, media pipelines and HPC systems execute in the cloud or on a cluster. Generic DAG engines close the section because they execute graphs in the cloud without an audio deadline. Before the families, the computation model itself is described.

### DSP DAG graphs as a computation model and their origin in DAWs

A DSP DAG is a dataflow graph with synchronous semantics. Each node consumes and produces a fixed number of samples per block, so the schedule of one block is static and can be computed once when the graph changes. This property is used by DAW engines, by the Web Audio API rendering graph, by GStreamer pipelines, and by domain-specific languages such as Faust that compile block diagrams to efficient code [2, 13]. The GPU-oriented API for DAWs proposed by Bianchi et al. formalizes the DAW project as a graph of tracks and effects and maps its independent branches to parallel GPU kernels, which confirms that the DAW graph is a suitable unit for parallel and distributed execution [9]. Real-time DAG scheduling theory studies the same structure for multicore embedded systems, including response time bounds along the critical path and the choice between global and partitioned scheduling [18]. Two quantities decide whether a distributed graph meets its budget: the block period, which is the block size divided by the sample rate, and the cost of the critical path, which is the sum of node processing times and network transfers along the longest path from a source to the output.

**Fig. 3** shows a small DAW session as a DSP DAG. Audio and MIDI clips are sources. Inserts such as compressors, equalizers, gates, and sound synthesizers are intermediate nodes. Sends route a copy of a signal to an auxiliary reverb bus. The sum bus and the master limiter are the last stages before rendering. The graph is acyclic, but it is not a tree, because sends create additional paths. The longest path from a source to the output defines the latency floor of the session. **Fig. 4** shows how such a graph maps to a microservice deployment. A gateway authenticates the client and enforces quotas. A graph compiler validates the graph, partitions it into subgraphs, and places them on node-type services. A block transport layer connects the services. Supporting services keep node state, store stems and renders, and collect observability data.

The DAW origin of the DAG matters for two reasons. First, the DAW already stores the graph, the plugin identifiers, the parameter automation, and the routing in its project file. A cloud audio engine can therefore obtain the DAG from the DAW instead of asking the user to rebuild it. Second, DAW plugins follow established formats such as VST, AU, CLAP, and, for the web, Web Audio Modules [2, 10]. A cloud node service that hosts such a plugin inherits its processing semantics, its parameter model, and its state model. This

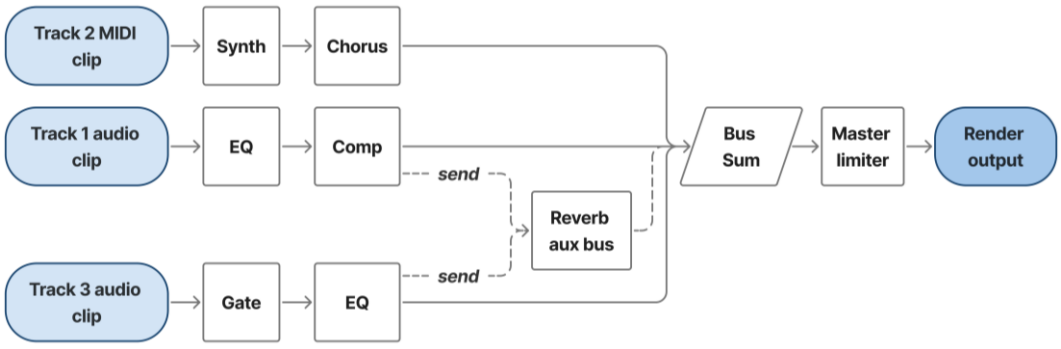


**Fig. 3.** A DAW session represented as a DSP DAG

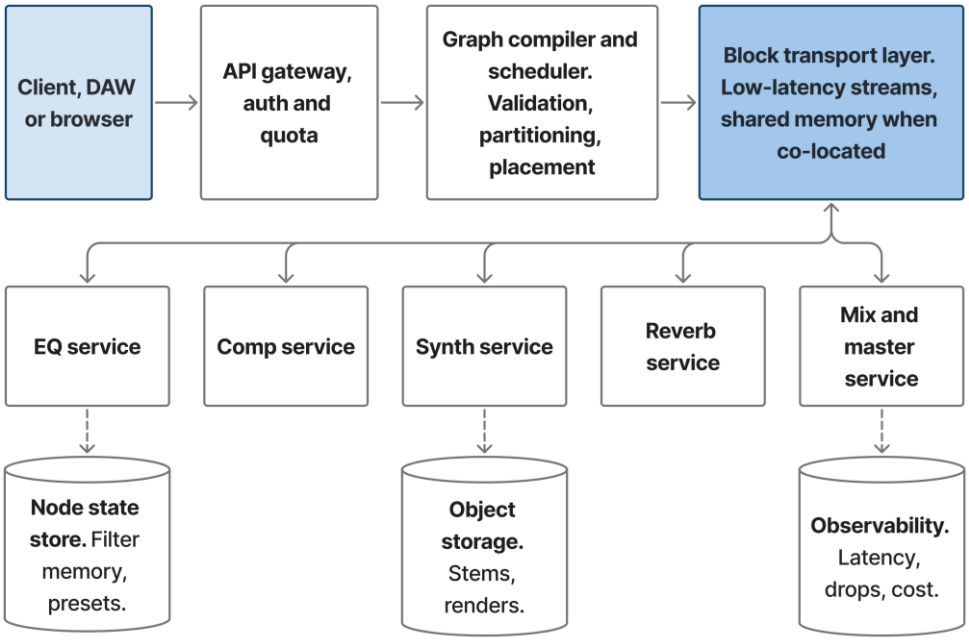


**Fig. 4.** A DAW session mapped to microservices

makes the plugin host the natural unit of a node service. It also brings licensing, sandboxing, and binary compatibility problems into the container, because most plugin formats were designed for a single desktop process.

Representation formats for audio objects and scenes exist at several levels, but none of them describes a processing graph. Turchet and Antoniazzi describe a semantic web of musical things that achieves interoperability between devices through ontologies and service descriptions [19]. Laitinen et al. present the OMASA format of the 3GPP IVAS codec, which jointly codes audio objects and a metadata-assisted spatial audio scene for immersive mobile communication at 13.2 to 512 kbps [20]. These formats describe devices, objects, and scenes, not DSP nodes with parameters and per-block state, which is what a cloud engine needs. Also, deep learning nodes are entering audio graphs. ANIRA is an architecture for neural network inference inside real-time audio applications that moves inference to a separate thread and respects the block deadline [21]. Such nodes are heavy and stateful, which makes them the first candidates for placement on remote services.

## Browser-based DAWs and Web Audio graphs

Browser-based DAWs are the closest existing systems to a cloud audio engine from the user point of view. WAM-Studio is an open-source DAW that uses Web Audio, Web MIDI, WebAssembly, and Web Components together with the Web Audio Modules plugin standard. Buffa and Vidal-Mazuy introduce it as a multitrack workstation for the web with plugin hosting and round-trip latency handling [16], and Buffa and Demont report on latency management, synchronized audio and MIDI recording, MIDI tracks, and offline rendering of a project [2]. The DAG in this system is the Web Audio rendering graph. Its nodes are native browser nodes and WebAssembly worklets. The graph executes on the client. The cloud stores projects, samples, and plugins and enables collaboration.

Other web-based systems use the same approach with a server that coordinates many browser clients. Matuszewski and Golvet describe rapid prototyping of distributed musical things with web technologies, where a server coordinates many browser clients that each run their own audio graph [22]. Buffa, Girard and Hofr use Web Audio Modules for immersive audio collaboration in the musical metaverse, where plugins run in every

client, and a server synchronizes the shared state [23]. Turchet and De Cet describe a web-based distributed system that integrates mobile devices of the audience in a choral performance [24]. Boem, Tomasetti and Turchet compare two-dimensional and mixed-reality interfaces for remote music making and find that usability, not immersion, decides adoption [25]. Gil Panal, David and Richard present Hi-Audio, an open-source web DAW for asynchronous multitrack recording and dataset distribution with a Flask REST backend, server-side annotation with machine learning models, server-side compression, and a browser round-trip latency estimation based on maximum-length sequences, whose variability reaches a standard deviation of about 8 ms in Chromium browsers [17].

The advantages of this family are zero installation, a standard graph model, a plugin standard, and mature low-latency execution on the client. The drawbacks are client device limits, the lack of server-side DSP execution, and the absence of a service model. Hi-Audio is the only system of the family with server-side processing, and it applies it to annotation and compression after upload, not to the audio graph [17]. Collaboration is implemented at the project level and not at the audio block level. The DAG is explicit, but it is not portable to a server-side engine because Web Audio nodes exist only in the browser. From the microservice point of view, these systems are monolithic clients with a thin cloud backend.

### Networked music performance and Internet of Sounds platforms

Networked music performance (NMP) systems connect musicians over a network. Turchet and Rottondi relate NMP to ubiquitous music and to the Internet of Musical Things and summarize the latency constraint of 20 to 30 ms mouth-to-ear delay [8]. Turchet and Tomasetti identify the latency contributions of immersive NMP systems, including capture, encoding, transmission, spatial rendering and playback, and relate them to available hardware and software [26]. Turchet and Casari assess a private 5G standalone and a public 5G non-standalone architecture for NMP and show which network configurations satisfy the latency budget [27]. The Internet of Sounds review by Turchet et al. places these systems in a broader agenda that includes cloud and edge processing, semantic interoperability, and sustainability [1]. Turchet and Casari also evaluate satellite links for NMP in rural areas, which extends the reach of such systems at the cost of higher and more variable delay [28]. Tomasetti and Turchet show that musicians prefer binaural audio with head tracking over stereo when playing with others through headphones, which adds a spatial rendering stage to the audio path [29].

NMP systems are relevant here because they set the latency budget for any cloud audio system that is used in a live setting. Their architecture, however, is a relay. A server or a peer forwards streams and, at most, mixes them. The DSP graph of each musician stays on the local machine. Packet loss concealment, jitter buffers, and clock synchronization are the main engineering topics. Vignati, Dallona and Turchet provide a modular testbench to study packet loss concealment algorithms [30]. Three works move the transport into hardware or dedicated radio to remove software jitter: an FPGA-based audio coprocessor [31], Ethernet audio transmission directly to an FPGA [32], and DECT-2020 NR radio for professional live audio [33]. The common finding is that the transport can be made deterministic when it is owned end to end, which a cloud service mesh does not offer. Real-time embedded platforms such as Elk Audio OS show that deep learning models can be executed inside the audio callback with deterministic latency, but again on a single device [34].

The advantage of this family is a well-studied latency model with perceptual thresholds and measurement methods. The drawback is that the DSP DAG is not executed remotely, so the results cannot be transferred directly to a cloud engine. Microservice principles appear only in the signaling and session management parts of these systems.

### Edge and MEC-based audio rendering

Edge computing brings compute close to the user and reduces the network delay of remote edges. Martusciello et al. implement a spatial audio service on a multi-access edge

computing (MEC) 5G infrastructure and analyze its performance [12]. The rendering stage runs on the edge server, and the client receives the rendered stream. The 3GPP immersive voice and audio services codec targets the same edge, using the OMASA format to carry separate talker objects along with a parametric scene captured by a mobile device, so rendering can happen at the endpoint or in the network at a low bitrate [20].

Fog-based orchestration extends this idea to many devices. Mikolajczyk et al. present gestural interaction with object-based audio in an Internet of Sounds ecosystem, where audio objects and their rendering are distributed across the ecosystem [35]. Vieira et al. formulate design guidelines for immersive environments that combine musical, multimedia, and multisensory things [36].

Remote production is the closest use case to a cloud audio engine in this family. Transporting many immersive channels to a remote mixing site is the main cost, which Giacomelli et al. document in their review of remote immersive audio production, along with the improvements they propose [4]. None of these works exposes the renderer as a graph of services.

The advantage of this family is the explicit trade-off between placement and latency, and the use of standard telecommunication infrastructure. The drawback is the granularity. The rendering stage is one service. The DAG inside the renderer is fixed and not exposed. Placement is static and chosen by the operator. Scaling to many concurrent sessions and fault tolerance are not evaluated, and only two of the three works in this family report a latency measurement. Still, these works provide the first measurements of rendering a DSP stage remotely with a network in the loop, which is exactly when network transfer enters the critical path.

### Microservice systems for music and audio tasks

A small number of works apply microservice architecture directly to music and audio functions. Raposo and Soares design a modular cloud architecture that adds generative music capabilities to a DAW through a VST plugin [10]. The backend consists of a gateway microservice with API key validation, a database microservice, a generative harmony microservice implemented in Java, and a chord-to-MIDI microservice implemented in Python, all exposed through Node.js REST services. The JUCE-based plugin runs inside the DAW, requests chord progressions, and drags the generated MIDI into the timeline. The measured computation time is about 865 ms for generation and about 1300 ms for MIDI conversion, and the network transfer time is negligible in comparison. The authors state that the latency is acceptable for composition but not for live performance, and that concurrency and containerized scaling remain future work.

Turchet and Krstulovic generalize this direction into a vision of DSP as a service (DSPaaS), in which musical devices connected to the Internet delegate production tasks to servers [11]. They identify the building blocks, the technological enablers and the network configurations of such a system, and they segment its applications into three tiers by latency constraint. Asynchronous services answer in seconds or more, reactive services within a few hundred milliseconds, and low-latency services within 10 to 30 ms or below, which is the threshold at which a musician no longer perceives the delay between an action and its sound. Three prototypes illustrate the tiers, among them a MIDI pattern recognition service in which a Raspberry Pi with Elk Audio OS sends recorded patterns over TCP/IP to a remote GPU server. The prototypes are analyzed under the key performance indicators of latency and reliability. The vision names the design trade-offs of a cloud audio service more precisely than any other reviewed work, but the server is treated as one service. Neither a processing graph nor a decomposition into services appears in it.

Work on microservices provides the architectural background for such systems. We proposed a framework for automated code scaling and execution on multiple distributed machines, where a primary service shares the user code with replica services and executes it across the network without an orchestrator [7]. An improved organization of REST API handlers that reduces project complexity and improves the maintainability of Node.js

microservices was proposed [6]. These results apply to the control plane of a cloud audio engine, that is, to gateways, session management, graph storage, and health monitoring.

The advantage of this family is a clean application of microservice principles, including gateways, independent services, and language heterogeneity. The drawback is that the services operate on requests and files, not on audio blocks. The DSP chain is absent because the services produce MIDI, code, or metadata, or it is hidden inside one server, and none of the systems executes a DAG with block-level deadlines. The three latency tiers of DSPaaS are the most useful classification of cloud audio services in the reviewed set, and only the third tier is a live audio graph problem [11]. The VST integration of Raposo and Soares is the only case where the DAW is the client of cloud microservices, and it uses the DAW as a user interface rather than as the source of the graph [10].

### Cloud native media processing pipelines

Media processing frameworks represent pipelines as graphs of elements with negotiated formats, which is the same model as a DSP DAG. Herranz Sanchez et al. present Voctomix 2.0, a cloud native architecture for real-time remote video production based on a GStreamer pipeline [13]. The mixing core, the ingest processes, and the operator interface are containerized. The architecture was evaluated on a native host, in a Docker Compose stack of eleven services, and in a single-node Kubernetes cluster. Containerization and orchestration added no appreciable processing overhead. The median command-to-output latency was 293 ms at 25 frames per second, which is determined by the frame period. Forced failures of a camera ingest were recovered in about 1.8 s by the restart policy, while a stream blanker kept the program output valid.

This work is the most complete evaluation of a containerized real-time media graph in the reviewed set. Three lessons transfer to audio. First, uncompressed internal traffic should stay inside a shared network namespace, because raw media between containers over a physical network is expensive. Second, self-healing by the orchestrator works at the scale of seconds, so output continuity needs a concealment mechanism in the graph itself. Third, telemetry should be decoupled from the processing path with asynchronous messaging. The drawback for audio is the time scale. A video frame period of 40 ms is an order of magnitude longer than an audio block period, and the audio path in this system is a single stereo stream that follows the video.

Related media systems confirm the pattern. Del Rio et al. deliver quality-of-experience-aware live media with a containerized processing chain for user-generated content [37]. Fukudome and Nishimura describe a cloud-based linear video delivery system that uses edge computing on a content delivery network [5]. Both systems operate at the frame or segment level. Neither reports an audio path with a block-level deadline.

### High performance and GPU acceleration for audio graphs

Turchet, Vella and Fiore analyze the potential of high-performance computing for the Internet of Sounds and propose a compute continuum where edge devices perform light feature extraction, network nodes aggregate, and HPC clusters with GPUs and FPGAs perform heavy rendering, such as spatial audio for thousands of concurrent users [38]. They identify the communication overhead between CPU and GPU and the deadline of the audio callback as the main obstacles for real-time use. Bianchi et al. show that a GPU-oriented API can execute the DAW graph in parallel on the GPU with acceptable latency, provided that the graph is analyzed and batched correctly [9].

The advantage of this family is throughput. A single accelerated host can render many graphs or very large graphs. The drawback is that the execution stays on one host and does not use a service model. Scheduling on the GPU is a task graph problem, and the results are directly useful for a node service that hosts an accelerated subgraph. The idea of a compute continuum also suggests a natural partitioning rule. Nodes with strict latency stay near the client, and nodes with heavy but tolerant computation move to the cluster.

### Generic DAG execution in cloud and serverless systems

Outside the audio domain, the execution of DAGs in the cloud is a mature research topic. Henning and Hasselbring benchmark the scalability of five stream processing frameworks deployed as microservices on Kubernetes, with more than 740 hours of experiments and up to 110 concurrent service instances [14]. All frameworks scale approximately linearly when resources are provisioned, but they differ in the rate at which resources must be added. Stream processing operators are stateful, run continuously, and communicate through message logs, which is closer to audio processing than request-based services. Their latency, however, is measured in tens of milliseconds to seconds and is dominated by the message broker. Fragkoulis et al. trace the evolution of stream processing systems and show that state management, exactly-once semantics, and elasticity are the dominant design goals, not bounded per-record latency [39].

Serverless platforms execute functions on demand and compose them into workflows. Raith, Nastic and Dustdar review serverless edge computing and identify placement, cold starts and stateful functions as open problems [15]. A cold start is incompatible with a block deadline of a few milliseconds unless the function is kept warm for the whole session. Real-time DAG scheduling theory, reviewed by Verucchi, Olmedo and Bertogna, provides response time analysis and schedulability tests for DAG tasks on multicore platforms under global and partitioned scheduling [18]. These results are formulated for a single machine with shared memory, so the network transfer delay between nodes is not part of the model. On the orchestration side, Dakic, Kovac and Slovinac evaluate Kubernetes for high-performance workloads with machine-learning-based placement [40]. Placement quality improves, but the scheduler does not see deadlines.

This family offers a rich set of scheduling, autoscaling, and benchmarking methods. The drawback is the absence of sample-accurate deadlines and of an audio transport. The frameworks optimize throughput and cost, whereas an audio graph needs bounded latency for every block. Still, the partitioning heuristics, the placement models, and the benchmark methodology of these works can be adapted to DSP DAGs.

### Comparison of the reviewed solutions

**Table 1** compares the eight solution families against the criteria defined in the Materials and Methods section. The families cover different points of the design space. Browser DAWs have an explicit DAG and a DAW origin, but no server-side execution. NMP and edge systems have real-time execution with a network in the loop, but no explicit DAG and no service decomposition. Audio microservice systems have service decomposition, but no DAG and no real-time mode. Cloud native media pipelines and generic DAG engines have both an explicit graph and an orchestrated deployment, but they lack the DAW origin and the audio block deadline. No reviewed system combines all four properties, namely an explicit DSP DAG, a DAW origin, block-level real-time execution, and a microservice deployment.

The measurement point column shows why the reported latency numbers are not comparable across families. Browser and NMP works report buffer or one-way network delays. Edge works report motion-to-sound delays. Microservice works report request processing times. The media pipeline reports command-to-output delay. There is no common benchmark, no common workload, and no common measurement point. This is a gap in itself, because a cloud audio engine must be evaluated along a defined critical path, with a defined block size and a defined network topology.

The reported figures of the cloud and container families, 293 ms command-to-output and 1.8 s recovery for the media mixer [13] and 0.8 to 1.3 s per request for the DAW microservice backend [10], lie two orders of magnitude above the 20 to 30 ms live budget of networked music performance [8], whereas the browser, NMP and edge families report values inside that budget but for a single device or a single stage. Because the cloud and container figures describe a request or a command and not a single audio block, this gap

*Table 1.* **Comparison of solution families for cloud, edge and browser-based processing of audio DSP graphs**

| Solution family | DAG location | Explicit DAG | Service granularity | Real-time mode | Reported latency | Measurement point | Scaling and fault tolerance | DAW origin |
|---|---|---|---|---|---|---|---|---|
| Browser DAWs and Web Audio graphs [16, 2, 23, 17] | Client | Yes, Web Audio graph | Monolithic client, cloud for storage | Block-synchronous on the client | Milliseconds | Audio buffer on the client | Limited by the client device | Native, web DAW |
| NMP and IoS platforms [8, 26, 27, 28] | Endpoints and relay server | No | Relay or session service | Streaming with 20 to 30 ms budgets | Tens of ms | Mouth to ear, one-way network | Network path, 5G slices, PLC | No |
| Edge and MEC rendering [12, 4, 20] | Edge server | Partly, one render stage | One rendering service | Real-time streaming | Reported on MEC [12] | End-to-end on the edge | Static MEC placement | No |
| Fog orchestration of audio scenes [35, 36] | Edge devices | Yes, scene model | Device level services | Soft real-time | Not reported | Not reported | Orchestrator assigns scene elements | No |
| Audio microservices and DSPaaS [10, 11, 7, 6] | Cloud | No, request pipelines | Task-level services behind a gateway, one server in DSPaaS | Offline, with reactive and low-latency tiers in DSPaaS | 0.8 to 1.3 s [10], tiers down to 10 to 30 ms [11] | Request round trip at the gateway, action to sound | Replicas, containerization planned [10] | Partial, VST plugin as client [10] |
| Cloud native media pipelines [13, 37, 5] | Containers on-premises or cloud | Yes, GStreamer graph | Pipeline per pod with sidecars | Frame synchronous | 293 ms, 1.8 s recovery [13] | Command-to-output frame, failure to recover | Restart policies, self-healing, blanker | No |
| HPC and GPU audio [38, 9] | Cluster or GPU host | Yes, GPU task graph | Not a microservice | Real-time on one host | Milliseconds | Audio callback on the host | Vertical, accelerator | Yes, DAW API [9] |
| Generic DAG engines [14, 39, 18, 15, 40] | Cloud and edge | Yes, workflow DAG | Function or operator per node | Best effort, no sample deadlines | Broker bound, cold starts | Broker end-to-end, function invocation | Autoscaling, retries, replay | No |

shows that these systems have not been measured against the per-block budget, not that they are unable to meet it. This observation answers RQ3.

### Microservice principles applied to DSP DAG execution

Microservice principles were formulated for business applications with request and response interactions. **Table 2** maps the main principles to the requirements of a DSP DAG and states how the reviewed works address each principle. Each principle is classified in the text as conflicting, neutral, or requiring an audio-specific interpretation. The mapping answers RQ2.

*Table 2.* **Microservice principles and their meaning for the execution of DSP DAG graphs**

| Microservice principle | Expectation in classic services | Requirement of a DSP DAG | Status in the reviewed work |
|---|---|---|---|
| Single responsibility | One business capability per service | One DSP node type or one subgraph per service, with the plugin host as the unit | Task-level services in [10], no node-level services |
| Independent deployment | Rolling updates without downtime | Updates must not change a running graph, node versions pinned per session | Not addressed |
| Statelessness | State in external stores, any replica serves any request | Filters, delays and reverbs hold sample state that must follow the stream | Session affinity implicit in [13], stateful operators in [14] |
| API contract | REST or gRPC request and response | Block streams with timestamps, ordering, and back pressure | Stream transport between containers only in [13] |
| Horizontal scaling | Add replicas behind a load balancer | Scale by the number of concurrent graphs, critical path is not shortened | Not tested for DSP graphs in [10] |
| Resilience patterns | Retries, timeouts, circuit breakers | Retries add latency, dropouts must be concealed within a block period | Self-healing in seconds [13], packet loss concealment in NMP [1] |
| Observability | Logs, metrics and traces per request | Per-block latency, missed deadlines, clock drift per session | Asynchronous telemetry in [13] |
| Orchestration | Kubernetes scheduling and autoscaling | Deadline-aware placement, co-location of chained nodes, warm instances | Real-time DAG scheduling theory only [18], cold starts as an open problem [15] |

Three principles conflict with DSP DAG execution directly. Statelessness conflicts with the internal state of DSP nodes. A replica cannot take over a filter without receiving its state, and the state changes every block. The practical answer is session affinity, where a graph or a subgraph is pinned to a service instance for the whole session, and failover is treated as a session migration with a short concealment. The request and response API contract conflicts with the streaming nature of audio blocks. A node service must expose a stream endpoint with sequence numbers, timestamps, and back pressure, and the gateway must not sit on the audio path. Resilience patterns such as retries and circuit breakers conflict with the block deadline. A retry that takes longer than one block period is a dropout. Instead, the graph must include concealment nodes, and the orchestrator must keep warm replicas so that failover fits into the latency budget.

Two principles are neutral and can be applied directly. Single responsibility maps to one service per DSP node type or per subgraph, which is the plugin host in a DAW-originated graph. Independent deployment maps to versioned node images that are pinned

per session, so that a rolling update of a node type does not change the sound of a running session. Three principles need an audio-specific interpretation. Horizontal scaling adds capacity for more concurrent graphs but does not reduce the latency of one graph, which is bounded by its critical path. Observability may be extended from request traces to block-level metrics such as missed deadlines and clock drift. Orchestration must become deadline-aware, so that chained nodes are co-located, and remote edges are placed only where the latency budget allows.

The choice of service granularity deserves a separate remark. One service per DSP node gives the finest reuse and isolation but adds a network hop for every edge, which is the worst case for the critical path. One service per subgraph, for example per track chain or per bus, keeps the hops on the critical path small and still allows independent scaling of heavy subgraphs such as convolution reverbs. One service per whole graph is a monolith per session and gives the lowest latency but no reuse of hot node types across sessions. The reviewed works do not evaluate this trade-off. The partitioning heuristics from generic DAG engines and the co-location rule from the media pipeline work are the starting points for such an evaluation [14, 13].

## Open problems and what is missing

The review answers the fourth research question with the following list of missing elements.

- Portable representation of DAW graphs. There is no open, engine-independent serialization of a DAW session as a DSP DAG with plugin identifiers, parameter automation, routing, and latency compensation. Web Audio Modules cover the browser [16, 2], GPU APIs cover one host [9], and the VST bridge covers only the user interface [10].
- Granularity model for stateful DSP nodes. No reviewed work defines how to partition a DSP DAG into services, how to pin subgraphs to instances, and how to migrate node state during failover within the latency budget.
- Deadline-aware orchestration. Container orchestrators schedule pods by resources, not by block deadlines. Real-time DAG scheduling theory does not include network hops [18], and serverless platforms suffer from cold starts [15].
- Low-latency block transport between services. Media pipelines keep raw traffic inside a shared namespace [13], and NMP systems optimize the network path [26, 27], but there is no service-to-service audio transport with timestamps, back pressure, and concealment that fits a microservice mesh.
- Concealment and clock discipline as graph elements. Dropouts caused by orchestration actions must be concealed in the graph, and all services of a session must share a clock. These functions are treated as network concerns in NMP and are absent in microservice works.
- Benchmarks and evaluation methodology. Reported latencies are not comparable across families. A benchmark with defined graphs, block sizes, topologies, and measurement points along the critical path is missing, similar to what exists for stream processing microservices [14]. The latency and reliability indicators of DSPaaS are a starting point for such a benchmark [11], and open multitrack material could come from platforms such as Hi-Audio [17].
- Plugin hosting in containers. Licensing, sandboxing, binary compatibility, and GPU access for desktop plugin formats inside containers are not studied, and the container security survey of Jarkas et al. shows that privileged hardware access widens the attack surface [41]. The compute continuum idea suggests a split between latency-critical nodes near the client and heavy nodes in the cluster [38], but no implementation exists.
- Cost and energy models. The elastic cloud model charges per resource time, whereas a session pins resources for its whole duration. The trade-off between per-session monoliths and shared node services is not quantified. Lostanlen and Bouchet warn that rebound effects make digital audio unsustainable when more

compute is offered without accounting for its energy [42], and Gabrielli, Principi and Turchet report case studies on sustainability in the Internet of Sounds [43].

Ranked by the number of included studies that touch each gap, low-latency block transport comes first with 10 studies, followed by the portable representation with 7, concealment and clock discipline and benchmarks with 5 each, deadline-aware orchestration with 4, plugin hosting with 3, cost models with 2, and the granularity model with none. The gap with no study is the one that a microservice-based engine cannot avoid. These gaps are connected. The representation defines the graph, the granularity model defines the services, the transport and the clock connect them, the orchestrator places them, and the benchmark evaluates the result. Progress on one element without the others will not produce a usable cloud audio engine. This is why the next step of our research is a reference architecture that covers all elements and a prototype that executes DAW-originated DSP DAGs as microservices with a measured latency budget.

## Answers to the research questions

RQ1. Eight solution families process or render audio as graphs. Browser DAWs, HPC and GPU systems, media pipelines and generic DAG engines use an explicit graph. NMP, edge and fog systems, audio microservices and the DSPaaS vision do not. Only browser DAWs and the GPU API originate from a DAW. The computation model is block-synchronous dataflow on the client and on single hosts, streaming with a budget in NMP and edge systems, and best-effort workflow execution in the cloud.

RQ2. Three principles conflict with block-synchronous execution. Statelessness, request-based API contracts, and retry-based resilience. Workarounds exist in the literature for two of them: session affinity and stream transport inside a shared namespace [13], and packet loss concealment from NMP [30]. Two principles apply without change: single responsibility and independent deployment. Three principles need an audio-specific interpretation: horizontal scaling, observability, and orchestration.

RQ3. The families report latencies at five different points. Browser DAWs report audio buffer delays on the client. NMP systems report mouth-to-ear or one-way network delays. Edge systems report motion-to-sound delays. Audio microservices report request round trips of 0.8 to 1.3 s. Media pipelines report command-to-output delays of 293 ms and recovery times of about 2 s, and serverless platforms add cold starts on top of every invocation. The figures are not comparable across families because they describe different quantities. The cloud and container figures lie two orders of magnitude above the 20 to 30 ms live budget, but they measure a request or a command rather than a single audio block, so they indicate that these systems have not been evaluated on the per-block path rather than that they cannot meet it.

RQ4. Eight elements are missing. Ranked by evidence, they are block transport, graph representation, concealment, and clock discipline, together with benchmarks, deadline-aware orchestration, plugin hosting, cost models, and the service granularity model, which no included study addresses.

## Validity challenges

Construct validity. The eight families and the seven criteria were defined by the authors. A different coding could merge the edge and fog families or split the generic DAG engines. The comparison of reported latencies is qualitative, because the papers measure at different points and no reviewed paper reports the transfer delay between two DSP nodes for audio blocks.

Internal validity. The screening and the extraction were performed by one reviewer. The Stage 2 record count is approximate because the publisher sites were searched through a general search engine and not through a database interface with an exportable log. The reported latencies were taken from the papers without re-measurement, and their measurement points differ, which is why **Table 1** records the measurement point next to each figure instead of comparing the figures directly.

External validity. 17 of the 43 included studies come from two proceedings volumes of one symposium, and a large share of them originates from a small number of European research groups. The full-text criterion excluded some work on DAG scheduling and serverless workflows, and 25 topically relevant papers, including edge rendering, fog orchestration, and audio microservice work that would have strengthened the three smallest families. Cloud gaming, WebRTC, and broadcast transport standards were covered by a small number of studies. The findings should be read as a characterization of the peer-reviewed literature in the selected venues, not of commercial practice.

Conclusion validity. The gap ranking counts citations, not the depth of treatment, and the counts are small. The statement that cloud and container latencies exceed the live budget by two orders of magnitude rests on a handful of reported figures from two papers. It is robust to the choice of block size, because a block of 256 or 512 samples changes the budget arithmetic by a factor of two or four while the reported figures exceed the budget by a factor of one hundred or more.

## CONCLUSION

This paper reviewed existing solutions for processing and rendering audio in the cloud as DSP DAG graphs with a focus on microservice architecture. 43 peer-reviewed studies were selected with an explicit protocol and grouped into eight solution families that range from browser DAWs, which execute an explicit DAW-originated graph on the client, to generic DAG engines, which execute graphs in the cloud without an audio deadline. The families in between move streams, single rendering stages, offline tasks, or frame-level media graphs to the edge or the cloud.

The comparison shows that no reviewed system combines an explicit DSP DAG, a DAW origin, block-level real-time execution, and a microservice deployment. The mapping of microservice principles to DSP DAG requirements shows that statelessness, request-based API contracts, and retry-based resilience conflict with block-synchronous audio, whereas single responsibility and independent deployment apply directly, and scaling, observability, and orchestration need an audio-specific interpretation. The comparison of reported latencies shows that the reviewed studies report different types of latency. Browser DAWs report the audio buffer delay on the client, networked music performance systems report the one-way network or mouth-to-ear delay, edge systems report the motion-to-sound delay, audio microservices report the request-response time at the gateway, and media pipelines report the command-to-output delay. These quantities are measured on different paths and with different workloads, so direct comparisons between families are difficult. The request-response and command-to-output figures of the cloud and container systems, from hundreds of milliseconds to seconds, lie two orders of magnitude above the 20 to 30 ms budget of networked music performance, but they describe the processing of a whole request or command and not the per-block path that this budget constrains. The figures therefore do not show that the existing systems fail to meet real-time audio requirements. They show that no reviewed cloud or container system has been evaluated against a per-block latency budget. A latency model that makes the reported figures comparable and a measurement of the transfer cost between chained node services are the subject of a companion article.

What such systems lack today is a portable representation of DAW graphs, a granularity model for stateful DSP nodes, deadline-aware orchestration, a low-latency block transport between services, concealment and clock discipline as graph elements, a common benchmark, plugin hosting in containers, and a cost model. Further research will address these elements with a reference architecture for microservice-based execution of DAW-originated DSP DAGs and with a prototype evaluated against the latency budgets established by networked music performance research.

## ACKNOWLEDGMENTS AND FUNDING SOURCES

The authors express their gratitude to the Ministry of Education and Science of Ukraine for support (Project No 0125U001883).

## COMPLIANCE WITH ETHICAL STANDARDS

The authors declare that they have no competing interests.

## AUTHOR CONTRIBUTIONS

Conceptualization, [O.C.]; methodology, [O.C., H.K.]; validation, [O.C., H.K.]; formal analysis, [O.C.]; investigation, [O.C.]; resources, [O.C., H.K.]; data curation, [O.C.]; writing, original draft preparation, [O.C.]; writing, review and editing, [H.K.]; visualization, [O.C.]; supervision, [H.K.]; project administration, [H.K.].

All authors have read and agreed to the published version of the manuscript.

## REFERENCES

[1] Turchet, L., Lagrange, M., Rottondi, C., Fazekas, G., Peters, N., Østergaard, J., Font, F., Bäckström, T., & Fischione, C. (2023). The Internet of Sounds: Convergent trends, insights and future directions. *IEEE Internet of Things Journal*, *10*(13), 11264–11292. https://doi.org/10.1109/JIOT.2023.3253602

[2] Buffa, M., & Demont, S. (2024). Can you DAW it online? The challenges of a web-based open source workstation. In *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)* (pp. 1–8). IEEE. https://doi.org/10.1109/IS262782.2024.10704163

[3] Dewey, C., Moore, A., & Lee, H. (2024). Practitioners' perspectives on spatial audio: Insights into Dolby Atmos and binaural mixes in popular music. *Journal of the Audio Engineering Society*, *72*(7/8), 504–516. https://doi.org/10.17743/jaes.2022.0153

[4] Giacomelli, S., Centofanti, C., Santos, J., Galbiati, M., Salvi, T., Graziosi, F., & Rinaldi, C. (2024). Remote immersive audio production: State of the art implementation, challenges, and improvements. In *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)* (pp. 1–10). IEEE. https://doi.org/10.1109/IS262782.2024.10704192

[5] Fukudome, D., & Nishimura, S. (2024). Cloud-based Internet linear video delivery system using edge computing on a CDN. *SMPTE Motion Imaging Journal*, *133*(6). https://doi.org/10.5594/JMI.2024/TPSA1162

[6] Chaplia, O., Klym, H., Popov, A. I., & Konuhova, M. (2025). Enhancing REST API handlers organization for Node.js microservices. *SN Computer Science*, *6*, Article 776. https://doi.org/10.1007/s42979-025-04311-8

[7] Chaplia, O., & Klym, H. (2023). Node.js framework for automated code scaling and execution on multiple distributed machines. In *2023 IEEE 18th International Conference on Computer Science and Information Technologies (CSIT)*. IEEE. https://doi.org/10.1109/CSIT61576.2023.10324052

[8] Turchet, L., & Rottondi, C. (2023). On the relation between the fields of Networked Music Performances, Ubiquitous Music, and Internet of Musical Things. *Personal and Ubiquitous Computing*, *27*(5), 1783–1792. https://doi.org/10.1007/s00779-022-01691-z

[9] Bianchi, D., Avanzini, F., Baratè, A., Ludovico, L. A., & Presti, G. (2022). A GPU-oriented application programming interface for digital audio workstations. *IEEE Transactions on Parallel and Distributed Systems*, *33*(8), 1924–1938. https://doi.org/10.1109/TPDS.2021.3131659

[10] Raposo, A. N., & Soares, V. N. G. J. (2025). Modular microservices architecture for generative music integration in digital audio workstations via VST plugin. *Future Internet*, *17*(10), Article 469. https://doi.org/10.3390/fi17100469

[11] Turchet, L., & Krstulović, S. (2024). DSP as a service: Foundations and directions. *IEEE Open Journal of the Communications Society*, *5*, 6212–6226. https://doi.org/10.1109/OJCOMS.2024.3464696

[12] Martusciello, F., Centofanti, C., Rinaldi, C., & Marotta, A. (2023). Edge-enabled spatial audio service: Implementation and performance analysis on a MEC 5G infrastructure. In *2023 4th International Symposium on the Internet of Sounds* (pp. 1–8). IEEE. https://doi.org/10.1109/IEEECONF59510.2023.10335480
[13] Herranz Sánchez, M., Llorente, Á., del Rio, A., & Jimenez, D. (2026). Beyond hardware mixers: A resilient cloud-native architecture for real-time remote video production. *Applied Sciences*, *16*(18), Article 9253. https://doi.org/10.3390/app16189253
[14] Henning, S., & Hasselbring, W. (2024). Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud. *Journal of Systems and Software*, *208*, Article 111879. https://doi.org/10.1016/j.jss.2023.111879
[15] Raith, P., Nastic, S., & Dustdar, S. (2023). Serverless edge computing—Where we are and what lies ahead. *IEEE Internet Computing*, *27*(3), 50–64. https://doi.org/10.1109/MIC.2023.3260939
[16] Buffa, M., & Vidal-Mazuy, A. (2023). WAM-studio, a digital audio workstation (DAW) for the Web. In *Companion Proceedings of the ACM Web Conference 2023 (WWW '23 Companion)* (pp. 543–548). ACM. https://doi.org/10.1145/3543873.3587987
[17] Gil Panal, J. M., David, A., & Richard, G. (2026). The Hi-Audio online platform for recording and distributing multi-track music datasets. *EURASIP Journal on Audio, Speech, and Music Processing*, *2026*, Article 26. https://doi.org/10.1186/s13636-026-00459-0
[18] Verucchi, M., Olmedo, I. S., & Bertogna, M. (2023). A survey on real-time DAG scheduling, revisiting the Global-Partitioned Infinity War. *Real-Time Systems*. Advance online publication. https://doi.org/10.1007/s11241-023-09403-3
[19] Turchet, L., & Antoniazzi, F. (2023). Semantic Web of Musical Things: Achieving interoperability in the Internet of Musical Things. *Journal of Web Semantics*, *75*, Article 100758. https://doi.org/10.1016/j.websem.2022.100758
[20] Laitinen, M.-V., Vasilache, A., Rämö, A., Paulus, J., & Eksler, V. (2025). Combined object-based audio and MASA format for enhanced spatial mobile communication. In *ICASSP 2025, 2025 IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 1–5). IEEE. https://doi.org/10.1109/ICASSP49660.2025.10889750
[21] Ackva, V., & Schulz, F. (2024). ANIRA: An architecture for neural network inference in real-time audio applications. In *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)* (pp. 1–10). IEEE. https://doi.org/10.1109/IS262782.2024.10704099
[22] Matuszewski, B., & Golvet, A. (2023). Rapid prototyping of distributed musical things using web technologies. In *2023 4th International Symposium on the Internet of Sounds* (pp. 1–5). IEEE. https://doi.org/10.1109/IEEECONF59510.2023.10335368
[23] Buffa, M., Girard, D., & Hofr, A. (2024). Using Web Audio Modules for immersive audio collaboration in the Musical Metaverse. In *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)* (pp. 1–10). IEEE. https://doi.org/10.1109/IS262782.2024.10704108
[24] Turchet, L., & De Cet, M. (2023). A web-based distributed system for integrating mobile music in choral performance. *Personal and Ubiquitous Computing*, *27*(5), 1829–1842. https://doi.org/10.1007/s00779-023-01709-0
[25] Boem, A., Tomasetti, M., & Turchet, L. (2025). Between immersion and usability: A comparative study of 2D and mixed reality interfaces for remote music making. *International Journal of Human-Computer Studies*, *203*, Article 103586. https://doi.org/10.1016/j.ijhcs.2025.103586
[26] Turchet, L., & Tomasetti, M. (2023). Immersive networked music performance systems: Identifying latency factors. In *2023 Immersive and 3D Audio: From Architecture to Automotive (I3DA)*. IEEE. https://doi.org/10.1109/I3DA57090.2023.10289169
[27] Turchet, L., & Casari, P. (2023). Assessing a private 5G SA and a public 5G NSA architecture for networked music performances. In *2023 4th International Symposium on the Internet of Sounds* (pp. 1–6). IEEE. https://doi.org/10.1109/IEEECONF59510.2023.10335359
[28] Turchet, L., & Casari, P. (2024). The Internet of Musical Things meets satellites: Evaluating Starlink support for networked music performances in rural areas. In *2024*

*IEEE 5th International Symposium on the Internet of Sounds (IS2)* (pp. 1–8). IEEE. https://doi.org/10.1109/IS262782.2024.10704207

[29] Tomasetti, M., & Turchet, L. (2023). Playing with others using headphones: Musicians prefer binaural audio with head tracking over stereo. *IEEE Transactions on Human-Machine Systems*, *53*(3), 501–511. https://doi.org/10.1109/THMS.2023.3270703

[30] Vignati, L., Dallona, S., & Turchet, L. (2023). PLC Testbench: A modular tool for the study and comparison of audio packet loss concealment algorithms. In *2023 4th International Symposium on the Internet of Sounds* (pp. 1–8). IEEE. https://doi.org/10.1109/IEEECONF59510.2023.10335357

[31] Bert, D., Domini, N., Peloso, R., Severi, L., Sacchetto, M., Bianco, A., & Rottondi, C. (2023). FPGA-based low-latency audio coprocessor for networked music performance. In *2023 4th International Symposium on the Internet of Sounds* (pp. 1–8). IEEE. https://doi.org/10.1109/IEEECONF59510.2023.10335333

[32] Cochard, P., Weber, J., Michon, R., Risset, T., & Letz, S. (2024). Ethernet real-time audio transmission to FPGA. In *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)* (pp. 1–7). IEEE. https://doi.org/10.1109/IS262782.2024.10704104

[33] Dürre, J., & Werner, N. (2024). DECT-2020 NR for professional live audio: Design space exploration and practical evaluation. In *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)* (pp. 1–7). IEEE. https://doi.org/10.1109/IS262782.2024.10704159

[34] Stefani, D., & Turchet, L. (2023). Real-time embedded deep learning on Elk Audio OS. In *2023 4th International Symposium on the Internet of Sounds* (pp. 1–10). IEEE. https://doi.org/10.1109/IEEECONF59510.2023.10335204

[35] Mikolajczyk, K., Trolland, S., Ilsar, A., McCormack, J., Ferguson, S., & Bown, O. (2023). Gestural interactions with object-based audio in an Internet of Sounds ecosystem. In *2023 4th International Symposium on the Internet of Sounds* (pp. 1–9). IEEE. https://doi.org/10.1109/IEEECONF59510.2023.10335408

[36] Vieira, R., Wei, S., Röggla, T., Muchaluat-Saade, D. C., & Cesar, P. (2024). Immersive Io3MT environments: Design guidelines, use cases and future directions. In *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)* (pp. 1–10). IEEE. https://doi.org/10.1109/IS262782.2024.10704141

[37] del Rio, A., Llorente, Á., Ortiz-Arce, S., Belesioti, M., Pappas, G., Muñiz, A., Contreras, L. M., & Christopoulos, D. (2025). Seamless user-generated content processing for smart media: Delivering QoE-aware live media with YOLO-based bib number recognition. *Electronics*, *14*(20), Article 4115. https://doi.org/10.3390/electronics14204115

[38] Turchet, L., Vella, F., & Fiore, S. (2023). The potential of high-performance computing for the Internet of Sounds. In *2023 4th International Symposium on the Internet of Sounds* (pp. 1–7). IEEE. https://doi.org/10.1109/IEEECONF59510.2023.10335270

[39] Fragkoulis, M., Carbone, P., Kalavri, V., & Katsifodimos, A. (2024). A survey on the evolution of stream processing systems. *The VLDB Journal*, *33*, 507–541. https://doi.org/10.1007/s00778-023-00819-8

[40] Dakić, V., Kovač, M., & Slovinac, J. (2024). Evolving high-performance computing data centers with Kubernetes, performance analysis, and dynamic workload placement based on machine learning scheduling. *Electronics*, *13*(13), Article 2651. https://doi.org/10.3390/electronics13132651

[41] Jarkas, O., Ko, R. K. L., Dong, N., & Mahmud, R. (2025). A container security survey: Exploits, attacks, and defenses. *ACM Computing Surveys*, *57*, 1–36. https://doi.org/10.1145/3715001

[42] Lostanlen, V., & Bouchet, L. (2024). Rebound effects make digital audio unsustainable. In *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)* (pp. 1–7). IEEE. https://doi.org/10.1109/IS262782.2024.10704126

[43] Gabrielli, L., Principi, E., & Turchet, L. (2025). Sustainability and the Internet of Sounds: Case studies. *IEEE Transactions on Technology and Society*, *6*(2), 165–180. https://doi.org/10.1109/TTS.2024.3513777

# ОГЛЯД МЕТОДІВ ХМАРНОГО ОПРАЦЮВАННЯ ЗВУКУ НА ОСНОВІ ОРІЄНТОВАНИХ АЦИКЛІЧНИХ ГРАФІВ ОБРОБЛЕННЯ СИГНАЛІВ У МІКРОСЕРВІСНИХ АРХІТЕКТУРАХ

***Олег Чапля, Галина Клим****
*Національний університет "Львівська політехніка",*
*вул. Степана Бандери, 12, Львів, 79013, Україна*

## АНОТАЦІЯ

**Вступ**. Ланцюги обробки аудіо у цифрових звукових робочих станціях (ЦЗРС, англ. DAW – digital audio workstations) є орієнтованими ациклічними графами (ОАГ, англ. DAG – directed acyclic graph) вузлів цифрового оброблення сигналів (ЦОС, англ. DSP – digital signal processing). Виконання таких графів у хмарних середовищах дає змогу забезпечити гнучкість обчислювальних ресурсів, можливість спільної роботи та доступ із термінальних пристроїв. Мікросервісна архітектура переважає у хмарних системах, але її принципи сформовані навантаженнями типу запит-відповідь, а не блоковою синхронною обробкою аудіо. У статті оглянуто наявні методи для хмарного оброблення та відтворення звуку, поданого у вигляді графів цифрового оброблення сигналів, та оцінено їх на відповідність принципам мікросервісного підходу.

**Матеріали та методи**. Застосовано систематичний протокол пошуку з явними критеріями включення до двох повних томів праць IEEE International Symposium on the Internet of Sounds, до пошуку в базах даних і на сайтах видавців та до зворотного відбору за списками літератури. З 97 праць симпозіуму та 48 повних текстів з інших джерел включено 43 рецензовані дослідження з доступними повними текстами, опубліковані з 2022 до 2026 року. Методи згруповано у вісім сімейств та оцінено за розташуванням, явністю графа, деталізацією сервісів, режимом реального часу, повідомленими затримками, механізмами масштабування та відмовостійкістю, а також підтримкою графів, що походять із цифрових звукових робочих станцій.

**Результати**. Браузерні ЦЗРС зберігають ОАГ на клієнті та використовують хмарне середовище для зберігання даних і спільної роботи. Системи мережевого музичного виконання ретранслюють потоки у межах жорстких бюджетів затримки, але не виконують офлайн-графів опрацювання звуку віддалено. Периферійні служби формування звуку переносять на периферію лише один етап. Мікросервіси виконують пакетні завдання через мережеві програмні інтерфейси. Хмарні медіаконвеєри виконують обробку звукових сигналів у контейнерах із самовідновленням, а універсальні рушії орієнтованих ациклічних графів планують і масштабують, але не гарантують точності на рівні окремих значень блоків. Жодна з оглянутих систем не виконує граф ЦОС, що походить із ЦЗРС, як набір мікросервісів із граничними термінами на рівні блоків. Затримки в оглянутих матеріалах виміряно в різних точках, тому їх неможливо порівняти. Хмарні та контейнерні системи повідомляють про кількість від сотень мілісекунд до кількох секунд.

**Висновки**. Наявним системам бракує переносного подання графів ЦЗРС, моделі подрібнення для вузлів цифрової обробки сигналів зі станом, планування за граничними термінами в засобах оркестрування контейнерів, передавання блоків між сервісами з малою затримкою та еталонних тестів хмарного формування звуку. Ці прогалини визначають програму досліджень хмарного опрацювання звуку на основі мікросервісів.

***Ключові слова***: хмарні обчислення, мікросервіси, цифрова обробка сигналів, орієнтований ациклічний граф, цифрова звукова робоча станція, надійність

| Received / Одержано | Revised / Доопрацьовано | Accepted / Прийнято | Published / Опубліковано |
|---|---|---|---|
| 08 September, 2026 | 18 September, 2026 | 21 September, 2026 | 28 September, 2026 |