SMPTE

MI

MOTION IMAGING JOURNAL

APPLICATIONS/PRACTICES

KEYWORDS SOFTWARE-DEFINED MEDIA FACILITIES // VIRTUALIZED MEDIA FACILITIES // DYNAMIC MEDIA FACILITIES

A era do hardware dedicado está sendo redefinida diante de nossos olhos, a prova disso é este artigo que propõe uma mudança de paradigma fundamental: A transição das rígidas arquiteturas síncronas para um *framework* de mídia assíncrono operando inteiramente sobre infraestrutura de TI padrão. Isso mesmo que você leu, eu disse padrão!

O artigo demonstra convincentemente por que devemos abandonar o processamento tradicional em *Lockstep* (um modelo de execução onde cada etapa do processamento ocorre em perfeita sincronia sequencial, comum em hardwares proprietários, mas ineficiente em CPUs modernas) em favor de sistemas flexíveis orientados a eventos. Essa mudança arquitetural é a chave para desbloquear o uso massivo de equipamentos COTS (*Commercial Off-The-Shelf*, termo que designa hardware e software padrão de mercado, como servidores e switches genéricos, disponíveis comercialmente) sem sacrificar a performance.

O autor demonstra como abandonar o processamento "*lockstep*" em favor de sistemas orientados a eventos pode empurrar as restrições de tempo real para as bordas ("*glass-to-glass*"), permitindo o uso eficiente de nuvem e COTS. Com análises que apontam uma redução teórica de 50% na latência e ganhos massivos em eficiência, esta leitura é obrigatória para quem deseja dominar o futuro das instalações de broadcast definidas por software. Vamos evoluir?

Tom Jones Moreira
(tvdigitalbr@gmail.com)

The New Paradigm of Software Architected Broadcast Facilities:

An Asynchronous Media Framework Running on IT Infrastructure

By Marwan Al-Habbal

Industries like high-frequency trading require time synchronization with massive amounts of data transported on networks, processed on CPUs and GPUs running on generic IT infrastructure. This is not different from what broadcast workflows require. There are benefits to extend the boundaries of where asynchronous processing is used today in broadcast to include the interconnect and push the real-time constraints to the edge.

Abstract

The technological innovations of Information Technology (IT) look compelling in addressing the challenges of market fragmentation and reduced budgets of the live broadcast industry. To enable the shift from bespoke hardware, a new paradigm of software broadcast infrastructure running on IT equipment, whether it be on-premises or in the cloud, is described in this paper. This asynchronous media framework bridges the gap between the requirements of broadcast and the asynchronous and distributed nature of IT—not only matching the capabilities of traditional hardware infrastructure but surpassing it by enabling the benefits of IT infrastructures. This paper covers the foundational concepts/comparisons of synchronous versus asynchronous operations. It explores system architecture in depth: framework design, media microservices, timing, and control. Additionally, the paper presents empirical measurements that highlight time savings achieved by processing streams asynchronously compared to real-time methods. Finally, it examines the implications of this new paradigm of software broadcast architecture for live broadcast environments.

Broadcast organizations today are faced with a complex conundrum: produce more content than ever before with fewer resources while maintaining broadcast-quality productions for market opportunities that are yet to be identified. They are tasked with architecting media facilities to adapt to this new reality. This presents a particularly difficult challenge since the technology stack traditionally required relies heavily on bespoke hardware infrastructure that must be acquired in advance, maintained, and amortized over multiple years.

Meanwhile, it is no secret that the past few decades in the broadcast industry have shifted from relying primarily on hardware solutions to employing more software-based options. Thanks to advances in Information Technology (IT), live TV production today is now powered in part by specialized software running on regular personal computers (PCs) with professional input/output (I/O) interfaces (like SDI or SMPTE ST 2110). While these advances have increased the flexibility and agility in operations,[1] the broadcast industry—specifically live media production—has not attained anything close to the level of agility attainable using IT systems. If made available in the context of broadcast requirements, this agility can respond to the challenges outlined. However, transitioning from traditional hardware-centric approaches to IT-based architectures presents challenges. Unlike broadcast, which relies heavily on clock-driven real-time signal synchronization, IT equipment operates in an event-driven, asynchronous manner. This necessitates a fundamental re-evaluation of how live video is managed and presents opportunities to create a new paradigm of software infrastructure that bridges the broadcast and IT worlds and meets broadcast pedigree for quality, latency, and redundancy. In this software infrastructure, all processing executes and interconnects significantly faster than in real-time, and real-time constraints are pushed to the edge with synchronous interconnects found only at the boundaries.

By fully embracing standard IT infrastructure, broadcasters can tap into the benefits of cloud computing and use it as the core of their live operations. Today's computing hardware and networks, whether on-site or in the cloud, offer impressive processing and data transfer speeds that allow for high-definition (HD) and ultra-high-definition (UHD) video to be processed faster than real time. While broadcast systems traditionally rely on synchronous methods, IT systems are asynchronous and distributed. What if this gap can be bridged to create top-tier live productions using standard IT setups, giving broadcasters access to the latest IT innovations for their daily needs while meeting their quality requirements, high availability, and low latency? A media software framework that acts as "broadcast infrastructure" running on IT equipment, described in this paper, reconciles the requirements with the capabilities. The key to achieving this is shifting from clock-driven or synchronous operations to timestamp-driven, asynchronous ones.

Synchronous vs. Asynchronous

Broadcast concepts are deeply rooted in synchronous thinking, which stems from the real-time nature of video. This section explores the core concepts and implications of synchronous and asynchronous processing and interconnects from the perspective of software-based applications.

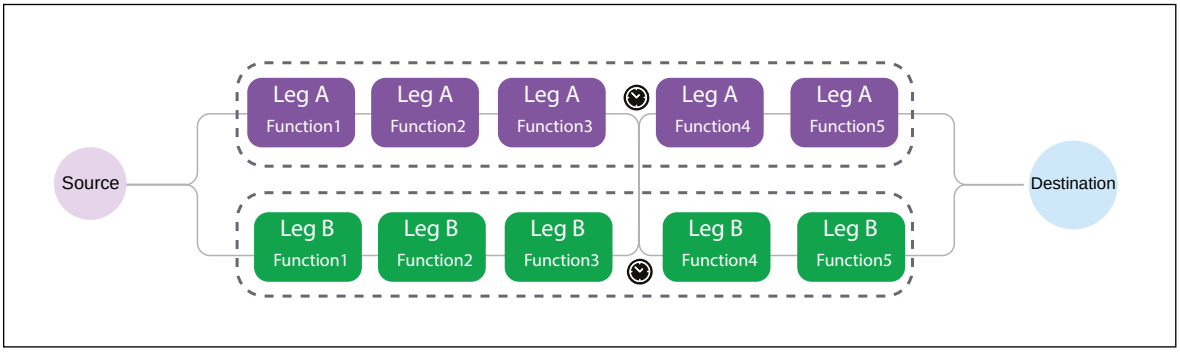

FIGURE 1. Clock-driven synchronous media facilities architecture.

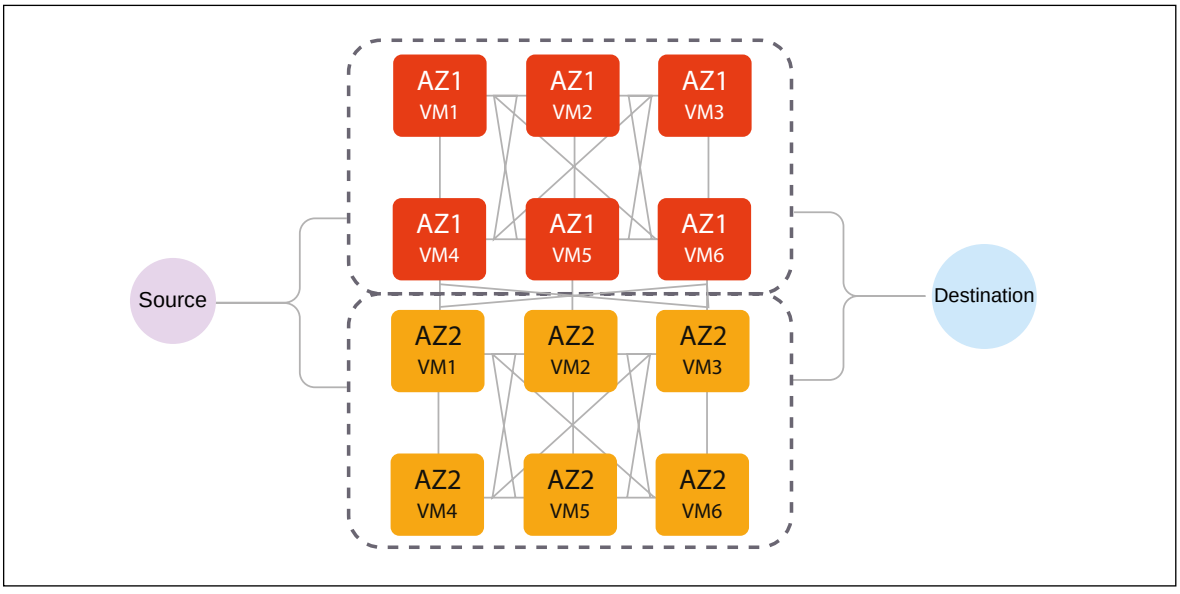

FIGURE 2. Event-driven architecture with interconnected mesh.

Underlying Principles

Synchronous operations or events occur coordinated and simultaneously, where each action is aligned with a common time reference. In synchronous systems, activities are performed in lockstep with one another—tied to one master clock. Live broadcast is synchronous—where baseband video, audio, and metadata are transported as a real-time signal. This is true whether it be legacy interconnects such as composite video or more modern interconnects such as 12G-SDI, Camera Data Interface (CDI), or SMPTE ST 2110. Each interconnect provides deterministic outcomes by delivering uncompressed pixels/samples to connect multiple processing chains throughout one frame. Each leg of a synchronous interconnect introduces a frame of delay **(Fig. 1).** Broadcast facilities are typically built with this architecture, where each function is linearly connected through a real-time interconnect, starting with a source and ending with a destination. Redundancy, if needed, is a time-locked replica that runs in perfect synchronization with the primary path.

In contrast, asynchronous operations are triggered by events and have no dependency on a common clock. Modern agile applications are built using event-driven architectures where events trigger decoupled microservices, complete their task as fast as possible, and communicate to other services by generating events themselves. Cloud-friendly architectures **(Fig. 2)** would have a distributed mesh where virtual machines (VM) are all interconnected across multiple machines with multiple availability zones (AZ), and the source and destination are not connected in a predetermined linear fashion.

Since clocks are not present in event-driven architectures, timestamping becomes essential to maintain real-time coherence when media is consumed on a screen or captured from a camera. The Joint Taskforce on Networked Media Reference Architecture[2] provides a framework for the media industry to transition to packet-based networks. It includes a model of grains with timestamps and identity. Using flows of grains, audiovisual content can be treated as data streams processed by general-purpose computers in software. This provides a viable alternative to explore.

Notably, the concept of event-driven architecture is not foreign to media facilities. They are microscopically found in all PC-based systems processing live media. In broad terms, a software application receives or transmits a synchronous signal through an I/O interface but processes it asynchronously with a Central Processing Unit (CPU) or a Graphical Processing Unit (GPU)—with a real-time window constraint. In other words, all software is already processed

TABLE 1. Maximum Number of Simultaneous Input and Output ST 2110-20 Flows Per Endpoint.

		Ethernet Bitrates					
		10 Gbit/s	25 Gbit/s	100 Gbit/s	200 Gbit/s	400 Gbit/s	800 Gbit/s
Resolutions (YCbCr 422 10 bit)	1920 x 1080i29.97	7 Rx 7 Tx	17 Rx 17 Tx	71 Rx 71 Tx	143 Rx 143 Tx	286 Rx 286 Tx	572 Rx 572 Tx
	1920 x 1080p59.94	3 Rx 3 Tx	8 Rx 8 Tx	35 Rx 35 Tx	71 Rx 71 Tx	143 Rx 143 Tx	286 Rx 286 Tx
	3840 x 2160p59.94	0 Rx 0 Tx	2 Rx 2 Tx	8 Rx 8 Tx	17 Rx 17 Tx	35 Rx 35 Tx	71 Rx 71 Tx
	7680 x 4320p59.94	0 Rx 0 Tx	0 Rx 0 Tx	2 Rx 2 Tx	4 Rx 4 Tx	8 Rx 8 Tx	17 Rx 17 Tx

as fast as the computer can run—with the processing throttled by its real-time inputs or outputs. Extending this concept beyond the boundary of a single computer is at the heart of this paper.

Scaling of Synchronous Interconnects

The generational improvements in baseband interconnects have brought forth lots of innovations. However, they have all remained real-time and synchronous. For example, the faster clock speeds of Serial Digital Interface (SDI) generations enabled higher-resolution transport. HD SDI at 1.485 Gbit/s can transport HD resolutions, whereas 12G-SDI at 11.88 Gbit/s can transport UHD resolutions.

On the other hand, standards like SMPTE ST 2110 have leveraged IP capability to offer more flexibility as a resolution-agnostic interconnect, where faster networks can carry higher resolutions and/or higher I/O density.[3] See **Table 1.**

Since video interconnects are bound to real-time, faster networks increase I/O throughput but do not decrease latency. Therefore, network speed increases allow for high resolutions/framerates or higher I/O density. Each case is examined separately below from an endpoint perspective.

In the first case of network speed evolution versus the need for higher resolutions/framerates, historical data shows that network speed evolution outpaces the need for higher resolutions/framerates. The IEEE 802.3 standard saw six Ethernet speeds added from 1980 to 2013, reaching 100 Gbit/s. By 2018, another six speeds were incorporated, boosting the maximum speed to 400 Gbit/s. This rapid progression continues, with ongoing efforts to develop 800 Gbit/s and 1.6 Tbit/s Ethernet.[4] This can be contrasted to the demand for higher resolutions. The release dates for SMPTE standards for transport demonstrate a much slower progression of the need for higher resolutions: SD SDI (270 Mbit/s—SMPTE 259M, 1989), HD SDI (1.485 Gbit/s—SMPTE 292M, 1998), 3G SDI (2.97 Gbit/s—SMPTE 424M, 2006) and 6G/12G SDI (11.88 Gbit/s—SMPTE ST 2082, 2015). Therefore, the higher speeds do not present mainstream benefits for an endpoint processing common broadcast resolutions.

In the second case of network speed evolution versus I/O density, each endpoint must process higher stream counts to maximize the available bandwidth utility. For example, an endpoint needs to process a total of 286 HD input and 286 output flows simultaneously to get the full benefit of 400 Gbit/s bandwidth. This fact incentivizes the development of monolithic and centralized architectures that process very dense I/O and are prone to bottlenecks and single points of failure. Therefore, the higher speeds do not provide practical benefits for an endpoint either.

In short, from an endpoint perspective, synchronous interconnects cannot provide meaningful benefits as network speeds increase. This is in contrast to High-Performance Computing (HPC) trends, where faster networking is used to create more agile environments, where finite hardware capacity is increased by scaling out and distributing the computational needs over the network.[5]

Theoretical Timing Diagrams

The processing chain of a live source can be broken down into processing/compute time and transport time. As described earlier, the processing done in software applications is asynchronous. It is triggered by receiving video frames at a clock boundary and throttled by a clock at its output. The implications of interconnecting synchronously vs. asynchronously can be explored through simplified theoretical timing diagrams of a basic workflow of a live camera feeding two cascaded PC-based mixers **(Fig. 3**). The black arrows signify a mandatory synchronous interconnect as the source or destination is "glass" (i.e., a camera or a monitor).

In the following analysis, the video payload is assumed to be 1920 × 1080 YCbCr 4:2:2 in 10 bits with a real-time processing window of 16 ms, a compute time of 4 ms per mixer, and 25 Gbit/s of interconnect (where applicable).

Two inefficiencies can be noted when examining the timing diagram using a synchronous architecture for the interconnects. **Figure 4** shows idle time when the mixers have completed their task for a given frame and compute resources are not utilized. The idle time lasts until the next video frame can be introduced at a subsequent frame tick. Furthermore, the transport latency introduces one frame

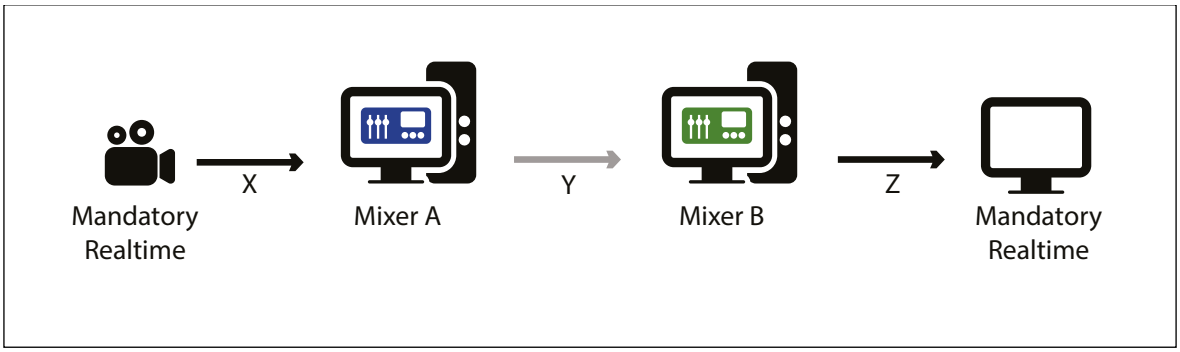

FIGURE 3. Simple setup for timing diagram analysis.

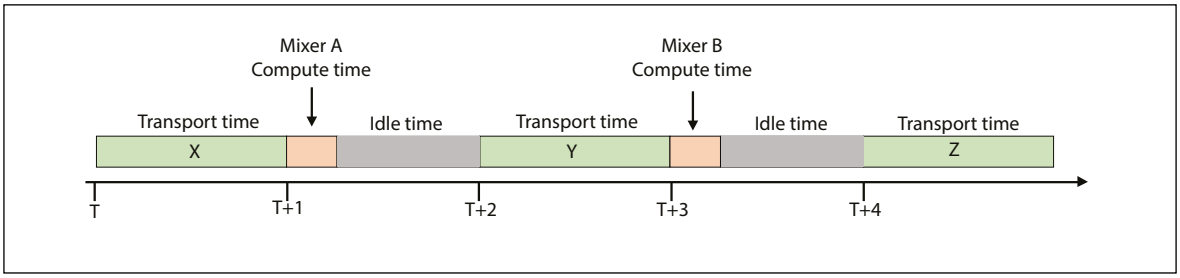

FIGURE 4. Theoretical timing diagram of a synchronous processing pipeline of two mixers.

per hop (where X and Z are mandatory, but Y is not).

For a source starting at time T, the overall output is available 4 frames later at T+4 with an overall latency of 64 ms. Time is split into 8 ms for compute, 32 ms for transport, and 24 ms for idle. See **Table 2** for a summary.

In this setup, idle time presents the system with two opportunities: increased reliability or additional computation. From a reliability perspective, idle time introduces a margin for each mixer process to be late for its task. In practice, this margin is desirable as momentary glitches can get absorbed by this idle time. From a computational perspective, on the other hand, idle time can be seen as additional time for each of the individual mixers to do more work. If not used for additional computation, idle time counts as wasted cycles. In other words, there is a computational incentive to build more monolithic processing for each step, where Mixer A and Mixer B would consume more time to accomplish additional work. Notably, the generational increase in hardware speed exacerbates this incentive to build monolithic processing. In the example above, the single task of mixing would take less time to complete on faster hardware while the frame boundary would remain where it is, thus increasing idle time.

TABLE 2. Time from In-to-Out for Synchronous and Asynchronous Interconnect Constructs.

	Synchronous Interconnect (Figure 4)	Asynchronous Interconnect (Figure 5)
Realtime Window	16 ms	16 ms
In-to-Out Delay	64 ms	32 ms
Transport Time (clocked)	32 ms	16 ms
Processing Time*	8 ms	8 ms
Idle Time	24 ms	6 ms
Asynchronous Transport (event-driven)**	N/A	2 ms

*Compute time for each mixer is estimated to be 4 ms

**Asynchronous transfer time is calculated for 10-bit 1920 x 1080 at a rate of 25 Gbit/s(1920 pixels/line x 1080 lines x 2.5 bytes/pixel x 8 bits/Byte) / (25 Gbit/s) = 1.7 ms ~ 2 ms.

In an event-driven architecture, Mixer A triggers the transport as soon as it completes its work, which triggers Mixer B to start processing. **Figure 5** shows that idle and transport times are reduced, resulting in higher efficiency and lower latency. For a source starting at time T, the overall output is available 2 frames later at T+2 with a latency of 32 ms, where time is divided into 8 ms of compute, 2 ms of asynchronous transport (Y), 16 ms of synchronous transfer (X) and 6 ms are idle. See **Table 2** for a summary.

With this pipeline, overall latency is reduced, and idle time is minimized. As discussed, idle time can also be further analyzed from two perspectives: increased reliability and additional computation. Unlike the synchronous processing analysis described earlier, idle time in this asynchronous model presents new modular opportunities.

From a reliability perspective, idle time can be seen as overall buffering for the combined vagaries in computing. This contrasts with the synchronous pipeline, where idle time, as buffering, is split per compute element. From a computational perspective, on the other hand, idle time presents opportunities to process video outside of the mixer processes for a more modular approach. In other words, maximizing hardware usage can still be achieved without

each mixer doing more within a monolithic architecture.

Note that the previous analysis is based on a theoretical construct of how data flows in a typical PC. In practical applications, additional buffering would be required between input and output as operating system (OS) variability exists in the handshaking between kernel and user space.

Despite being theoretical constructs, the concepts described give insight into the impact of synchronous interconnects on software application design. **Table 2** summarizes the results side by side, showing that within this simplified model **(Fig. 3)**, the overall latency is reduced by 50%. In comparison, idle time is reduced by 75% (or equivalently, efficiency is increased by 75%). Empirical measurements of the latency of an asynchronous system are presented in a later section.

Key Insights

Other industries like high-frequency trading (HFT) require time synchronization while dealing with massive amounts of data transported on fast, low latency networks, processed on CPUs and GPUs, all using event-driven architectures running on generic IT infrastructure. This is not different from what broadcast workflows require. As demonstrated earlier, there are compelling benefits to extending the boundaries of where asynchronous processing is used today (within the bounds of a single PC) in broadcast to include the interconnect and push the real-time constraints to the edge—as close to "glass" as possible. Synchronous interconnects implicitly enable core operational control requirements and are extremely important technologies when real-time transport is mandatory. However, they introduce undesirable consequences (as described in the previous section) for a software broadcast facility—meant to run on generic IT equipment.

Asynchronous Media Framework

What is required to make the broadcast, and more specifically, the handling of live media, operate efficiently at scale on generic IT infrastructure, whether on-premises or in the cloud? The next section presents an asynchronous media framework that makes this possible while providing the broadcast pedigree for robustness, frame accuracy, quality, low latency, and best-of-breed choices.

Asynchronous Processing Paradigm

A modern media facility can benefit from IT innovations by leveraging event-driven rather than clock-driven architectures with asynchronous rather than synchronous processing.

Figure 6 shows an asynchronous approach to media processing for live production. Streams arrive and leave in real time. Sequences of frames (grains) are received through the glass of a lens and are displayed through the glass of a screen. Each uncompressed grain in every input stream is given an identity based on its capture timestamp. Similarly, each output stream consists of grains with a presentation timestamp. Uncompressed grains are placed in a content fabric and processed asynchronously, as fast as possible, across one or more systems.[6]

Media Services

A media service is a specialized, high-quality, uncompressed media-processing capability implemented in software as a process. **Figure 7** shows the context for media services within an asynchronous framework.

In the framework, services are written to a relatively simple interface. The framework fires an event and asks the service to produce a frame for a given timestamp. The service then asks for control information for that point in time. It

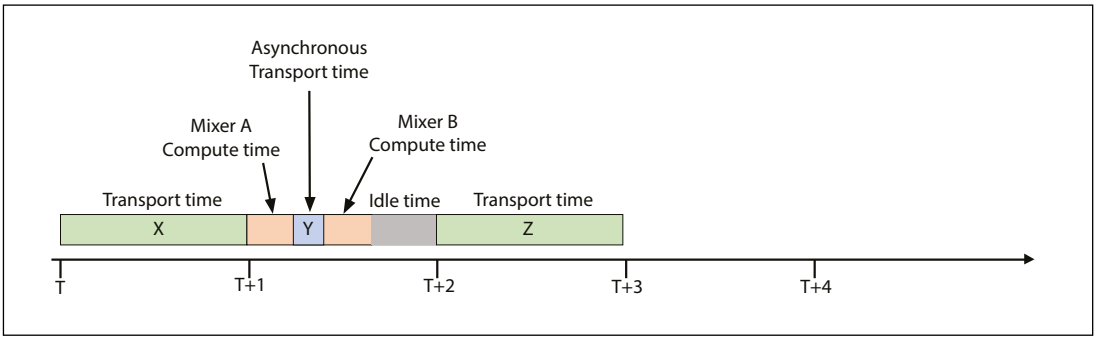

FIGURE 5. Theoretical timing diagram of an asynchronous processing pipeline of two mixers.

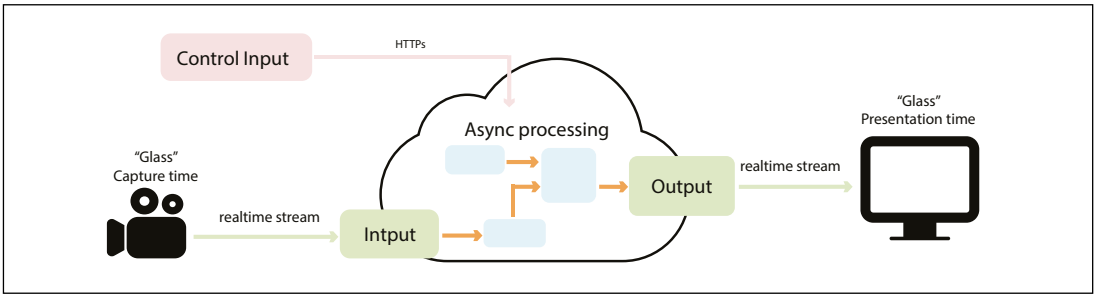

FIGURE 6. Glass-to-glass asynchronous processing.

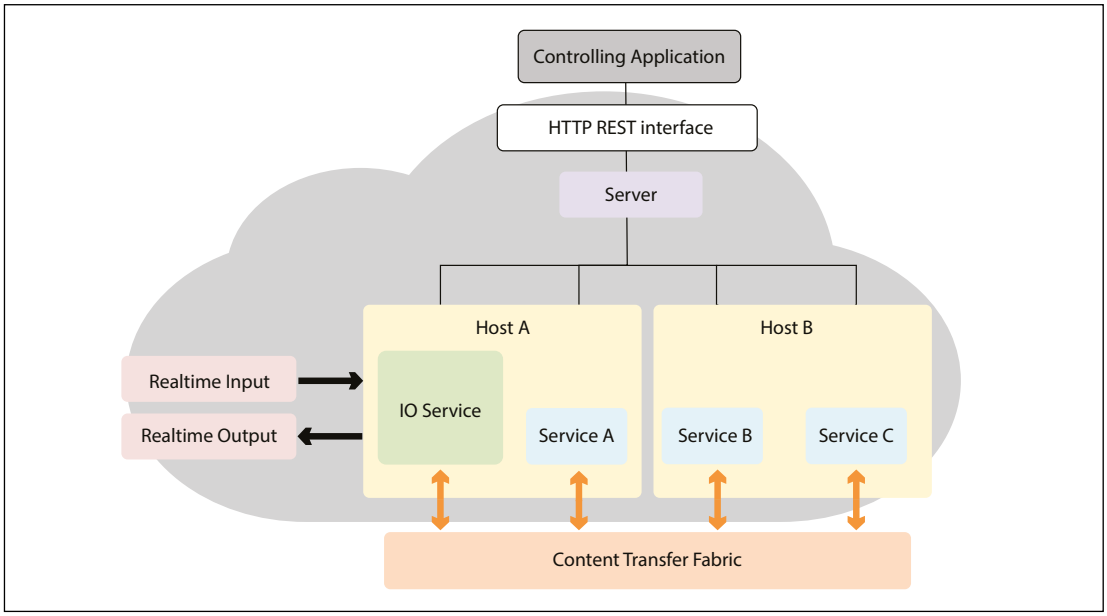

FIGURE 7. Media services in an asynchronous media framework.

uses the control values in the response to request the media sources it requires further. Once all the service's requests are satisfied, it produces and writes its output. With the framework defining the media formats and control model, a media service is an isolated component of media processing logic.

How, where, and when the media service runs, and how many copies of the service exist at any given time, is controlled by a server. The server manages the lifecycle of hosts, where each host includes a shared memory pool and coordinates work on a single computer. The server also manages the instance lifecycle of media services. Each service has zero-copy access to the local memory pool. If a service on one host needs to consume frames produced on another host, a link is established, and grains flow across the content transfer fabric.

Multiple replicas of the same service can be distributed across two or more hosts, each replica doing the same work. If one replica fails, the others still do the work, and no frames are dropped. In this way, the complexity associated with scaling, migrating, and protecting against failure is the responsibility of the framework. It is not a concern for the implementer of a media service. The control application of the framework needs only to decide to create a replica of a media service and to execute it where it makes sense. The framework manages the seamless transfer of grains when and where needed.

Real-time inputs and outputs come in and out of the framework via I/O services, which can provide multi-path hitless protection by having multiple replicas distributed across hosts. Input services assign an identifier to each grain. Usually, this will be based on the real-time at which the grain is presented to the input service, or it might be derived from timing data embedded in the signal (such as timecode). Output services retrieve grains from the fabric and deliver them in real-time using a timestamp derived from the grain identifier. Connections to SDI and SMPTE ST 2110 are possible where suitable hardware is available on-premises.

Control Tracks

A core concept of the asynchronous media framework is its control model, which is common to all media services and called the control track. Treated as just another kind of track alongside audio, video, and ancillary data tracks, control tracks configure the processing done by a media service at a given point in time (or, more generally speaking, provide a timeline of control). The state of the control track is maintained in the server and queried by the media service for each grain. The result is that media service implementations can be stateless. The data type of each control track is arbitrary and is defined by the author of the media service that originates the stream. Examples of control tracks include a playlist containing a schedule of clips to be rendered by a file player service, the transparency of a layer on a mixer, etc. The control model targets timestamped grains, allowing for frame-accurate control—independent of a predefined signal path. This is in contrast to traditional synchronous systems where control targets time, and time is associated with a real-time signal being transported along a well-defined clocked signal path.

Media Routing

One special type of control track is a router destination, which specifies which media tracks a media service should read to compose its output. The values placed on this control track define the routes of grains: from inputs, through any number of intermediate processing steps, to outputs. This enables the framework to be used as a frame-accurate, non-blocking, distributed clean-switching router, in which any grain can be routed anywhere—fully independent of what is traditionally known as a signal path.

Content Transfer Fabric

The content transfer fabric acts like a high-performance key-value store of grains. It is built using shared memory and accelerated networking. Its primary function is to deliver grains when and where they are needed. A media service requests a grain from the fabric with an ID, which remains blocked until the content transfer fabric delivers that grain. As soon as the requested grain is made available, a media service can complete its task as fast as possible (as described in its control track for that specific grain). If a requested grain is available locally, the content transfer fabric will provide this by referencing shared memory. If the desired grain is not available locally, the content transfer fabric will transport the grain over the network. The content transfer fabric is built using a network plugin architecture. This allows designing the most suitable networking technology according to the target hardware platform. Other industries using computers and networks—both on-premises and in the cloud—transcend the data rate requirements of a broadcast facility. These same technologies can be used for accelerated networking within the content transfer fabric.

Some examples of accelerated networking technologies include Elastic Fabric Adapters on Amazon Web Services (AWS) and network cards supporting Remote Direct Memory Access (RDMA). The Open Fabrics Alliance[7] also creates advanced networking ecosystems. They have provided *libfabric*,[8] a software library that enables developers to gain maximum performance and efficiency from whatever underlying network is available. Leveraging these technologies, the content transfer fabric can move data and saturate the network within and across availability zones.

Observability

Operational viability requires that observability and monitoring be embedded directly into the framework. From an operational perspective, workloads need to be correctly orchestrated, problems that may occur need to be diagnosed and tracked, and preventative maintenance on systems that are not working optimally needs to be undertaken.

These operational requirements are especially true as the computing hardware and the network are shared resources. Basic metrics on hardware utilization, such as CPU, memory, and network usage of all hosts, become mandatory. Furthermore, the health status of all the software processes implementing the transfer fabric, server, and media services must also be monitored. Additional higher-level metrics that measure the processing time used by the fabric and the media services are also required to understand the feasibility/reliability of a defined workload. All this information is transported and centralized into the server, where additional services can process the relevant information to transform it into a desired format, such as visual indicators or alerts.

Empirical Latency Results

Using the Asynchronous Media Framework described earlier, latency was measured on systems running in the public cloud. The results were obtained by running the Asynchronous Media Framework on AWS EC2 (c6in.8xlarge) instances that provide up to 50 Gbit/s of network bandwidth. The tests were run on interlaced video 1920 × 1080i25 streams with five cascaded mixers connected to a live source.

Figure 8 depicts the three test cases used to measure the latency impact of asynchronous processing and transport: all mixers on one host, two hosts within the same availability

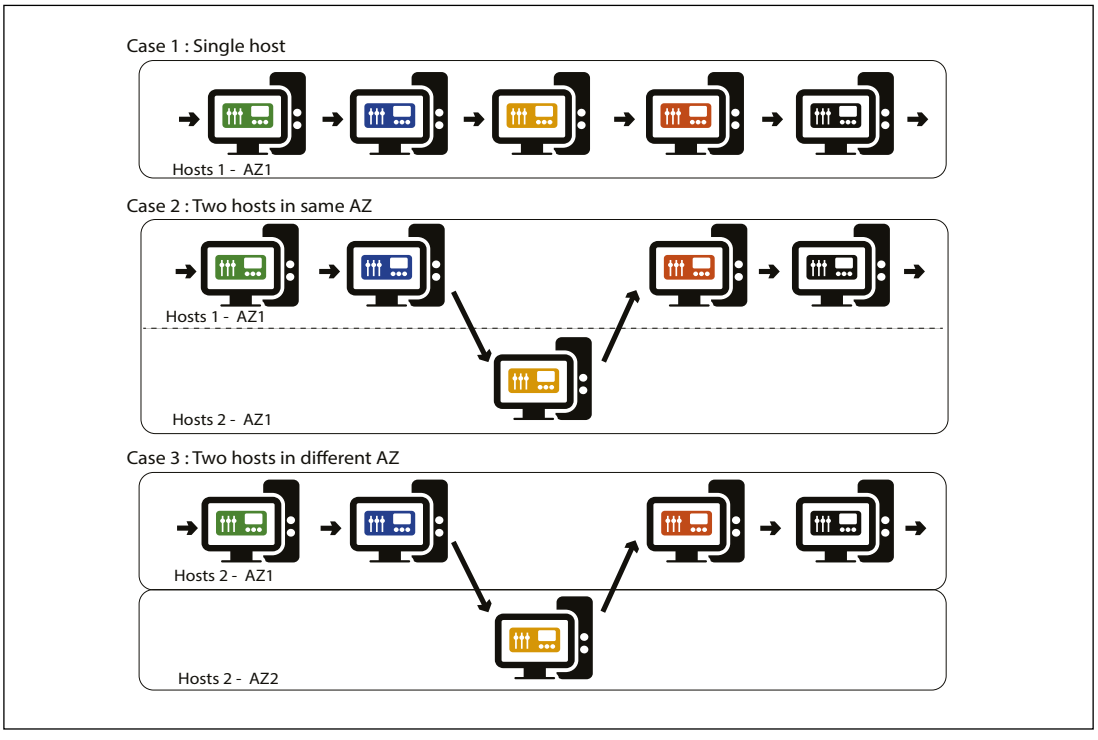

FIGURE 8. Test cases running on c6in.8xLarge AWS EC2 instances for latency measurements.

TABLE 3. Latency Measurements of Test Cases Using Asynchronous Media Fabric.

	Theoretical Reference	Case 1	Case 2	Case 3
Setup	Synchronous Interconnect	Single host	2 hosts same AZ	2 hosts different AZ
Latency	>200 ms	~6 ms	~15 ms	~26 ms

zone, or two hosts in different availability zones. Availability zones place up to 100 KM[9] between each mixer media service. The latter test is not a practical architecture, but it is useful to understand the impact of additional networking overhead. All these are compared to the baseline of five mixers connected with a real-time interconnect. The baseline model's latency is expected to be at least ten frames (the timing diagram is similar to **Fig. 4**).

Table 3 summarizes the latencies incurred in each of the test cases. Case 1 is the most efficient compute layout, as all data is accessed through shared memory. It executes 33.3x faster than a synchronous reference. Case 2 introduces some networking overhead but is at least 13.3x faster than a traditional synchronous approach. Case 3 introduces a lot of networking overhead, where uncompressed video is transported in and out of availability zones to cover <200 KM. It still provides a latency gain of at least 7.7x compared to the synchronous reference.

These tests demonstrate that the latency in an event-driven architecture is a fraction of the time required to perform the same task as a synchronous implementation.

Industry Initiatives

Industry organizations have been interested in proposing architectures and seeking solutions to achieve low latency, interoperable communication, and control of software media functions over the years. Some of these include the EBU Framework for Interoperable Media Services (FIMS),[10] EBU Media Cloud and Microservice Architecture (MCMA),[11] Open Services Alliance for Media,[12] Cisco Herisson,[13] and more recently, the EBU Dynamic Media Facility (DMF)[14] and the Video Services Forum Ground-Cloud-Cloud-Ground Working Group API.[15] These initiatives highlight the prolonged interest in solving this challenge and further validate the need for the proposed paradigm of software-architected broadcast facilities.

Conclusion

Many industries have already taken advantage of cloud computing to reap IT benefits like scalability, reliability, agility, and composability. Moving away from the traditional synchronous approach to broadcasting and designing media facilities using standard IT infrastructure can apply these same benefits to dynamic live production workflows, which can easily migrate between on-premises and cloud environments. However, clocked interfaces provide operational functionality such as deterministic outcomes and the ability to control large-scale systems. To replace these capabilities within a complete IT environment, an asynchronous media framework, as described, fills this gap. Clocked interfaces remain extremely important in this new paradigm but are refocused for usage at real-time interfaces projected from a camera or to a monitor.

The Asynchronous Media Framework removes the tight reliance on clocks while still offering frame-accurate control and low-latency responsive operations that are reliable and redundant. Next-generation software media facilities can be built to run on generic IT infrastructure using event-driven concepts rather than clocked-broadcast-specific concepts. As described, these next-generation software media facilities yield higher compute efficiency, lower latency, and better overall reliability compared to traditional PC-based deployments.

By providing APIs for custom development, the Asynchronous Media Framework allows users to license solutions from preferred vendors or code their own to tailor their media facilities to their needs. Agile, software-based broadcast setups can be launched at the press of a button—connecting media services to create customized, resilient workflows that can be easily scaled or shut down as needed. This software infrastructure can scale to fit various needs, from small productions with just one commentator (for self-operation) to large-scale setups involving multiple operators or automated systems. These agile software broadcast setups running on top of on-demand computing make it possible to base production decisions on business opportunities rather than the limitations imposed by technology.

As they continue to discover the benefits of software-architected broadcast facilities, developers, systems integrators, and broadcast engineers will need to tackle emerging challenges, such as creating specialized media processing, customizing automation and control systems, building user interfaces, and optimizing DevOps—all tailored to their specific needs. Furthermore, as the broadcast industry keeps evolving, its adoption of IT technology creates enormous growth potential. A software framework that allows best-of-breed software applications to work seamlessly together on standard IT infrastructure—without sacrificing key broadcast production requirements—will not just boost broadcasters' agility, flexibility, scalability, and operational capabilities but will also set the stage for an exciting future promising new and novel innovations in broadcast technology.

Acknowledgment

The author would like to acknowledge the work of Martin Lafferty, who invented the asynchronous media frame-

work. This framework, as described, is the culmination of a career pioneering large-scale broadcast automation systems.

References

1. Jean Lapierre and Marwan Al-Habbal, "Bridging the Gap Between Software and SMPTE ST 2110," presented at the SMPTE 2018 Annual Technical Conference & Exhibition, Los Angeles, CA. Oct. 2018. ISBN: 978-1-61482-960-7.
2. Joint Task Force on Networked Media Reference Architecture. [Online]. Available: https://www.jt-nm.org/reference-architecture
3. SMPTE, ST 2110-20:2022, "Professional Media Over Managed IP Networks: Uncompressed Active Video."
4. Institute of Electrical and Electronics Engineers (IEEE) Standards Association. [Online]. Available: https://standards.ieee.org/beyond-standards/ethernet-50th-anniversary/
5. Neil Ashton and Chethan Rao. "Elastic Fabric Adapter: Enabling Highly Scalable HPC & ML Workflows on AWS." [Online]. Available: https://www.youtube.com/watch?v=2LX5JnbCbhQ
6. Richard Cartwright, "Asynchronous Software for Live Production" EBU, presented at the Network Technology Seminar, Geneva, Switzerland, May 2023.
7. OpenFabrics Alliance. [Online]. Available: https://www.openfabrics.org
8. LibFabric OpenFabrics. [Online]. Available: https://ofiwg.github.io/libfabric
9. Amazon Web Services. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/aws-fault-isolation-boundaries/availability-zones.html
10. European Broadcast Union. *Framework for Interoperable Media Services* (17 Sep 2014). https://tech.ebu.ch/publications/ebu_tech_fs_fims
11. European Broadcast Union. *Media Cloud and Microservices Architecture*. https://tech.ebu.ch/groups/mcma
12. Open Services Alliance. [Online]. Available: https://www.smpte.org/rapid-industry-solutions/osa
13. Cisco Herisson. [Online]. Available: https://github.com/cisco/herisson
14. European Broadcast Union. *The Dynamic Media Facility Reference Architecture* (2024-09-03). [Online]. Available: https://tech.ebu.ch/publications/white-paper-2024-09-03
15. Video Services Forum (2025). *Ground-Cloud-Cloud-Ground (GCCG) Transport API*. [Online]. Available: https://github.com/vsf-tv/gccg-api

About the Author

Marwan Al-Habbal is a product manager at Matrox Video, where he develops hardware and software enabling broadcast solutions in IT environments. He holds an Electrical Engineering degree and MBA from McGill University.