Analysis of the development approaches of the system of audio synthesis and recognition with the option of using photonic processors

Voitko, Viktoriia, Bevz, Svitlana, Burbelo, Sergii, Stavytskyi, Pavlo, Khoshaba, Oleksandr, et al.

Viktoriia V. Voitko, Svitlana V. Bevz, Sergii M. Burbelo, Pavlo V. Stavytskyi, Oleksandr M. Khoshaba, Natalia O. Rysynets, Olena Yu. Teplova, Andrzej Smolarz, Saule Smailova, Assel Mussabekova, Bakhyt Yeraliyeva, "Analysis of the development approaches of the system of audio synthesis and recognition with the option of using photonic processors," Proc. SPIE 12040, Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments 2021, 120400N (3 November 2021); doi: 10.1117/12.2611464

Event: Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments 2021 (WILGA 2021), 2021, Warsaw, Poland

Analysis of the development approaches of the system of audio synthesis and recognition with the option of using photonic processors

Viktoriia V. Voitko*[a], Svitlana V. Bevz [a], Sergii M. Burbelo [a], Pavlo V. Stavytskyi [a], Oleksandr M. Khoshaba [a], Natalia O. Rysynets [a], Olena Yu. Teplova[b], Andrzej Smolarz [c], Saule Smailova [d], Assel Mussabekova [e], Bakhyt Yeraliyeva[f]

[a]Vinnytsia National Technical University, Khmelnitsky highway, 95, 21021,Vinnytsia, Ukraine; [b]Vinnytsia State Pedagogical University named after Mikhail Kotsyubinsky, Vinnytsia, Ukraine; [c]Lublin University of Technology, Nadbystrzycka 38A, 20-618 Lublin, Poland; [d]East Kazakhstan State Technical University named after D.Serikbayev, Ust-Kamenogorsk; [e]Academy of Logistics and Transport, Almaty, Kazakhstan; [f]M.Kh.Dulaty Taraz Regional University, Taraz, Kazakhstan

ABSTRACT

Approaches to the development of the system of audio synthesis and analysis are considered. Described ready to use cloud solutions and their drawbacks that can make engineers choose custom implementation in order to satisfy required use cases that are to be implemented. Considered an option of using optic fiber processors on the server-side application in order to optimize the music recognition flow. Considered various options of build systems that help to configure project build configuration which is especially important while developing cross-platform solutions. Therefore, technologies for developing cross-platform applications and systems are also described. In order to make the system work efficiently, it is important to build all the components in a correct way in order to provide a clean implementation and architecture. Additionally, there are considered approaches of server-side optimization of music recognition and search process by introducing load balancing and parallel matching.

Keywords: music recognition, multiplatform, cross-platform, mobile, database search, sharding, optic fiber processor

1. INTRODUCTION

When developing software like the system of audio synthesis and recognition [1] it is important to pay attention both to the algorithmic part and the implementation of client and server components. The solution developed should be reliable and contain as little duplicate code as possible. This will ensure the high quality of the solution and increased developer productivity. In order to do this, it is important to consider the technologies that allow cross-platform development including mobile platforms. Moreover, it is required to ensure the high speed and stability of the server-side that works on music recognition and matching. That is why it is required to consider technologies, approaches and algorithms that allow achieving this goal and provide a decent user experience.

2. RESEARCH RESULTS

The easiest way to implement a software system that recognizes music is to use cloud services that provide similar functionality as an API. One of the examples of such software is ACRCould. It includes a developed music recognition solution that provides an API for the client applications. It also includes a preconfigured database of music fingerprints. In order to use such a solution, there should be integrated SDK or used an API by a client application and all the implementation and algorithmic parts will be solved by the service vendor.

One of the advantages of such an approach is that most of the implementation is already available to the client applications. The engineer is required to integrate an API and handle the business cases that his application should implement and the user interface. Therefore, it is possible to achieve a pretty native integration of music recognition functionality to the developed application. However, this approach has a disadvantage. Such an approach does not achieve enough flexibility if it is required by the application. The source code of the SDK and the server-side is closed

*dekanfki@i.ua

sourced which means that engineers are not able to analyse or modify it. Some of the use cases require specific modification or optimization for a smooth user experience. For example, one of such scenarios is the ability to play music with voice for its further editing and music tracks creation.

When developing a custom solution the developer has full control of the software and its settings and is able to set up the technology for the specific use cases. On the other hand, such an approach requires more engineering efforts and resources and hence requires a deep analysis of the possible approaches and technologies that allow achieving the most efficient result.

The implementation of the music recognition component should consist of two main parts. Firstly, the input audio stream should be sampled using the Fast Fourier Transform algorithm. This will allow the input signal to be prepared for further transformation. Next, an audio fingerprinting hash algorithm should be implemented for efficient and optimised music matching [1].

One of the experimental options that can be applied on the server-side of the system is using an optical fiber processor. Such processors use photons produced by lasers for computation that allow higher bandwidth and performance than the electrons. Moreover, optic fiber processors could be integrated as part of traditional computer systems. Since for music recognition flow the system uses Fourier Transform for audio signal sampling, this operation can be executed by such a processor. The main additional step is to convert the input audio signal into a visual representation. This is required to use laser beams and lenses in order to perform a computation [2, 3, 4].

One of the most important parts of the entire music recognition system is the client-side. It is important to have an opportunity to reuse common source code on various platforms such as Android, iOS, Web and desktop operating systems. One of the goals behind that is an efficient integration of music recognition algorithms by avoiding rewriting them in different languages. This will help to provide decent system stability and avoid inconsistency between client applications developed for each of the available platforms. Also, it will help to reduce the number of errors and bugs in source code [5].

In any case, it is impossible to avoid using multiple programming languages when developing such a software solution. Each of the client platforms has its own features and specifics of working with hardware, the architecture of which might vary. Therefore, even with using cross-platform technologies, there will still be some part of source code that should be developed natively. That is why it is important to analyze the advantages and disadvantages of available build systems in order to select the one that will manage the project build configurations in the most efficient way.

The default build system on the Android platform is Gradle. It is a flexible build system that allows precise configuration using a build DSL based on Groovy or Kotlin languages. It is possible to create custom Gradle plugins that can extend its functionality and describe special build rules and tasks based on required use cases. On the other hand, Xcode is the default build system for Apple platforms. In order to develop a server-side solution, the build system will differ depending on the selected programming language and ecosystem.

One of the available cross-platform solutions is KotlinMultiplatform which allows having a common source code both for Android and iOS platforms. Moreover, it provides an ability to have a common code with the server-side in case the latter is written in Kotlin. Such an approach is possible to implement due to using the Gradle build system and appropriate plugins that implement multi-platform functionality. The user interface source code should be native to the specific platform in this case. This means that it will be implemented separately for each platform. However, engineers have an opportunity of reusing the command business-logic code that can include implementations of use cases algorithms etc.

Another approach is to use Flutter. Flutter is the technology that allows developing cross-platform solutions using Dart programming language. It includes Android, iOS, Web and desktop platforms. One of the core differences between Flutter and Kotlin Multiplatform is that the former focuses on making source code for the user interface common. This means that the UI for all the platforms should be written once and reused by all platforms. Moreover, common logic could be also implemented the same way. Only platform-specific code should be separated [2,5,6].

One more approach is to use native platform code in order to implement the user interface. However, common logic might be implemented using C++ or Rust languages. In this case, developers can use the Bazel build system as one of its advantages is the fact that it can build projects written in multiple languages. One more benefit that this technology brings is built granularity, which means using smaller code modules that can be reused in parallel in order to speed up the build time.

Considering the cross-platform approach and the nature of the music recognition component of a system of music synthesis and analysis it is important to develop an appropriate architecture and component structure in order to ensure stability and high performance of the system. The communication between client applications and the server-side should be considered. Moreover, the architecture should have an abstraction layer for the platform-specific implementations of the user interface and communication with hardware APIs (fig. 1) [4].

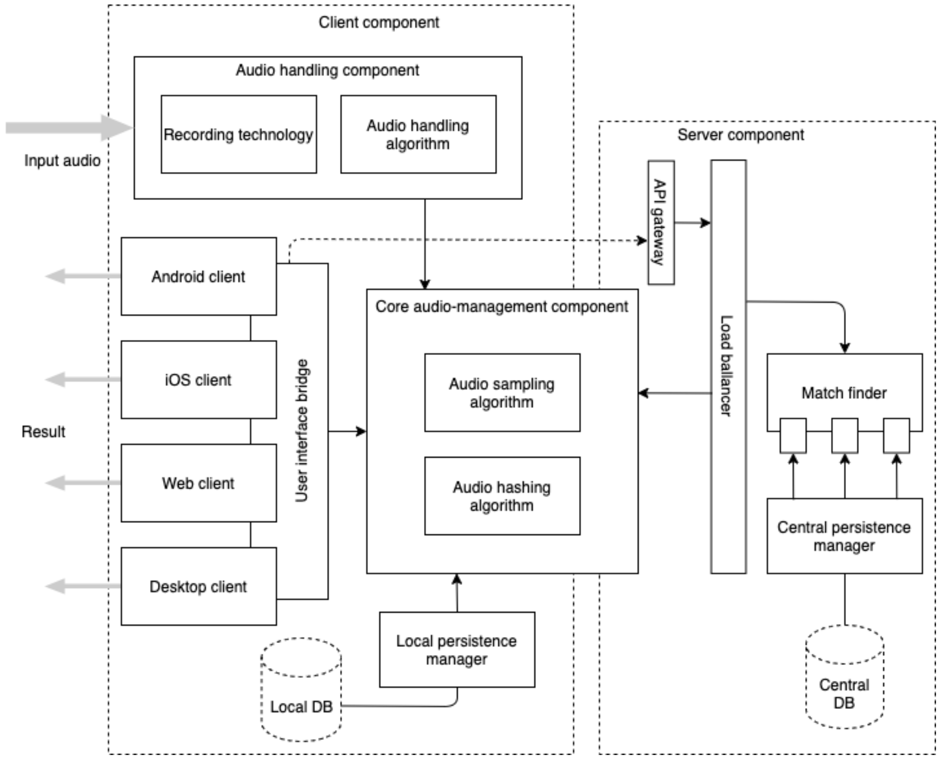

Figure 1. Music recognition system structure

As displayed in figure 1, the developed system consists of the server-side and the client-side. Both of them are using core audio-management components which stands as a primary building block of the music recognition system. This module is common for both of the parts in order to be able to perform a remote music recognition using the server-side the same as the client does. This component includes audio sampling and audio hashing and fingerprinting algorithms that form the core of the music recognition flow. This helps to achieve high-speed recognition and work in environments with a high noise level which is not a part of the music composition itself [1, 6, 8].

On the client side, the system receives the input audio data using the appropriate module which takes the audio input from the device microphone. Implementations of such modules are platform-specific as they use platform-specific hardware APIs in order to receive the input audio streams. On Android, there can be used technologies such as AAudio and OpenSL ES, while on iOS it is AVFoundation API.

It is important to implement local storage for client applications that can implement caching features and increase the application performance in some cases.

When the audio input is recorded and transformed by the client application it sends it further to the server using HTTPS protocol for further analysis and matching.

In order to manage server loads that can be caused by multiple client requests, there should be a load balancing system that makes the load on various system clusters approximately equal. When the server receives the request with the audio fingerprints it passes to the audio analysis module. The central database should already contain a set of prepared fingerprint hash values of different audio tracks. It is crucial to implement the process of parallel fingerprint searching and matching as this is the process that helps to identify the metadata of the requested song. There are multiple ways to achieve this [9,10,11,12].

One of such approaches is database sharding that performs horizontal partitioning of the database [6, 7]. It allows splitting the database into multiple parts where each of them contains a specific segment of the overall dataset. Such data can be grouped based on different contexts that help to efficiently perform parallel queries to the database.

Moreover, it is required to handle quickly the data received from the database. Fingerprint matching can be performed using parallel CPU computing. When working with parallel task execution there should be considered a degree of parallelism that can be calculated using formula 1.

$$DOP = PARALLEL\_THREADS\_PER\_CPU * CPU\_COUNT * INSTANCE\_COUNT \tag{1}$$

where *DOP* denotes the degree of parallelism; *PARALLEL_THREADS_PER_CPU* – the number of execution threads that CPU can handle; *CPU_COUNT* – the number of CPUs and *INSTANCE_COUNT* is the number of parallel segments of a database cluster.

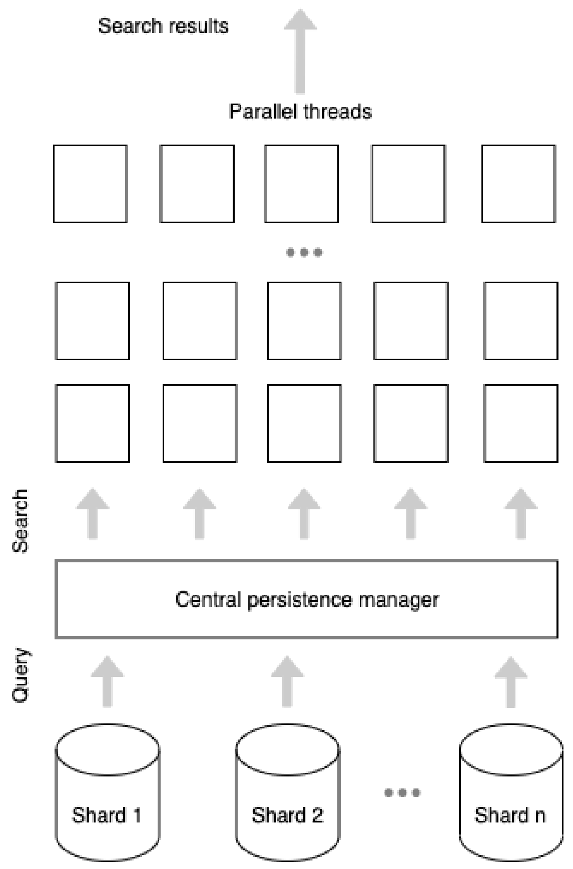

Figure 2. The process of parallel fingerprint matching search

However, even server-side hardware will not allow enough degree of parallelism in order to efficiently perform a large number of small tasks such as music fingerprint matching. On the other hand, GPU is exactly what is needed in order to achieve efficient execution of such a task. The execution can be split into a large number of small tasks and ran on a large number of parallel threads. Such an approach allows increasing the overall speed of the fingerprint matching process. As an example, such an approach is applied in technology called CUDA. Its parallel execution model is represented in formula 2.

$$NOPT = X\_BLOCK\_DIM * Y\_BLOCK\_DIM * X\_THREAD\_DIM * Y\_THREAD\_DIM * Z\_THREAD\_DIM \tag{2}$$

where NOPT denotes the number of parallel GPU threads; X_BLOCK_DIM is the thread block size by OX axis; Y_BLOCK_DIM is the thread block size by OY axis; X_THREAD_DIM is the thread grid size by OX axis inside a thread block; Y_THREAD_DIM is the thread grid size by OY axis inside a thread block; Z_THREAD_DIM is the thread grid size by OZ axis inside a thread block.

Therefore, by executing tasks on a GPU it is possible to achieve parallel execution on multiple thousands or even hundredsthousand of threads. This can significantly improve the speed of parallel music fingerprint matching process. The process of parallel fingerprint matching search is displayed on a figure 2.

In order to implement such an approach, there can be used NVIDIA CUDA technology that supports C++ and Fortran programming languages. One of the disadvantages of selecting this particular technology is that it supports only NVIDIA GPUs. However, if appropriate hardware is available there can be achieved a significant speed boost.

3. CONCLUSION

Different approaches to implementing the music recognition component in the system were analysed and a custom solution was chosen that would allow more flexibility when developing the different features of the system. Modern technologies enabling cross-platform development were also analysed, which will help to make common source code with reusable algorithms on different platforms without introducing duplicates, inconsistencies and additional bugs. By using Flutter, it is possible to implement a single user interface and business logic that can be reused across multiple platforms. Flutter also has a built-in build system that encapsulates native solutions. Furthermore, implementing parallel search and matching approaches will help increase the overall speed of music recognition. It is also important to consider ways to optimise computation using the optical processor to increase overall system performance considerably. Therefore, all the components reviewed will help implement a high-performance music recognition component in a music synthesis and analysis system and provide a decent user experience.

REFERENCES

[1] Voitko, V. V., Bevz, S. V., Burbelo, S. M., Stavytskyi, P. V., Pinaiev, B., Omiotek, Z., Baitussupov, D., and Bazarbayeva, A., “Automated system of audio components analysis and synthesis,” Proc. SPIE 11045, 110450V, (2019). https://doi.org/10.1117/12.2522313

[2] Delia, L., Galdámez, N., Thomas, P., Corbalan, L., Pesado, P., “Multi-platform mobile application development analysis,” IEEE 9th International Conference on Research Challenges in Information Science (RCIS), 181-186 (2015). https://doi.org/10.1109/RCIS.2015.7128878.

[3] Mikheenko, L.A. and Mikitenko, V.I., “Energy calibration of multispectral high-resolution scanning devices,” Space science and technology 15(3), 42-49 (2009).

[4] Mikheenko, L.A. and Borovitsky, V.N., “Variable brightness emitter based on conjugated integrating spheres,” Technology and design in electronic equipment 6 (66), 61-64 (2006).

[5] Wang, A.L., “An Industrial Strength Audio Search Algorithm,” Proceedings of the 4 th International Conference on Music Information Retrieval, (2003). https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf

[6] Annamalai, M., Ravichandran, K., Srinivas, H., Zinkovsky, I., Pan, L., Savor, T. and Nagle, D, “Sharding the Shards: Managing Datastore Locality at Scale with Akkio,” 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), 445-460, (2018).

[7] Varichenko, L.V., Kolobrodov, V.G., Ladyka, Ya.E., Mikitenko, V.I. and Mikheenko, L.A.. “Methods and means of measuring the energy characteristics of optical-electronic systems of space sensing of the Earth,” Space science and technology 2/3(12), 59-69 (2006).

[8] Pavlov, S.V., Kozhemiako, V.P., Kolesnik, P.F., Kozlovska, T.I., Dumenko, V.P., “Physical principles of biomedical optics,” NTB, Vinnitsa (2010).

[9] Oujezsky, V., and Horvath, T., “Traffic analysis using netflow and python,” Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska, 7(2), 5-7 (2017). https://doi.org/10.5604/01.3001.0010.4823

[10] Pavlov, S.V., Vassilenko, V. B., Saldan, I. R., Vovkotrub, D. V., Poplavskaya A. A. and Kuzin O. O., “Methods of processing biomedical image of retinal macular region of the eye,” Proc. SPIE 9961, 99610X (2016). https://doi.org/10.1117/12.2237154

[11] Grzechca, D., Pelczar, P. and Chruszczyk, Ł., “Analysis of Object Location Accuracy for iBeacon Technology based on the RSSI Path Loss Model and Fingerprint Map,” Intl. Journal of Electronics and Telecommunications 62(4), 371-378 (2016). https://doi.org/10.1515/eletel-2016-0051

[12] Rovira, R., Bayas, M. M., Pavlov, S.V., Kozlovskaya, T.I., Kisała, P., Romaniuk, R.S., and Yussupova, G., “Application of a modified evolutionary algorithm for the optimization of data acquisition to improve the accuracy of a video-polarimetric system,” Proc. SPIE 9816, 981619 (2015). https://doi.org/10.1117/12.2229087