# A system to generate speech to text in real time

Kavyashree Shankar
Illinois Institute of Technology
Chicago, US
kshankar1@hawk.iit.edu

Saptarshi Chatterjee
Illinois Institute of Technology
Chicago, US
schatterjee@hawk.iit.edu

***Abstract*— A real time speech to text conversion system converts the spoken words into text . Speech-to-Text technology enables us to convert audio to text by applying powerful neural network models. It has number of applications for users with and without disabilities. Speech-to-text has been used for voice search, help writers boost their productivity, and to provide alternate access to a computer for individuals with physical impairments. Other applications include speech recognition for foreign language learning, voice activated products for the blind and many familiar mainstream technologies. It is a driving force behind the success of new age voice-controlled speakers like Amazon Echo and Google Home.**

**While cloud solutions like Google Cloud Speech-to-Text, Amazon Transcribe, IBM watson Speech to Text etc effectively convert audio and voice into written text , but there are some scalability challenges in effectively using them. In this project we tried to address some of those challenges and also tested what are the configurations and parameters yield best results for such a system.**

*Keywords*— Speech-to-Text, voice search, Amazon Echo, Google Home

## 1. Introduction.

In this article we are proposing a scalable system architecture to generate text from speech input in real time. We also are publishing our experiment results and performance metrics for different technologies used as the building blocks of this architecture.

## 2. Related Work, Challenges And How We Addressed These Issues.

A lot of work is already been done in this field. Cloud services became fairly good at converting audio and voice into written text using Machine Learning Algorithm like Neural Network , but still such a system has scalability challenges in effectively using them. Following are the few challenges that drives most of the architecture decisions while designing such a system -

### *A. Selection of scalable system design patterns.*

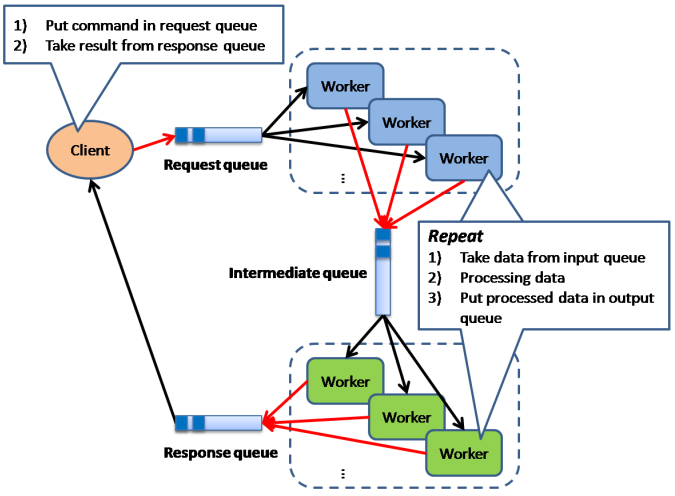
If we push streaming audio directly to web server, then such a system is bound to choke under heavy load, as CPU will be busy processing previous inputs and new inputs will be waiting to be consumed and eventually system will start throwing 5XX errors . So we need to provisionally store the audio file , before an available process can pick it up and start processing .

We researched on several Scalable System Design Patterns[2] and found out " event-based architecture" model is most suitable to address this problem .Event-based architecture supports several communication styles:

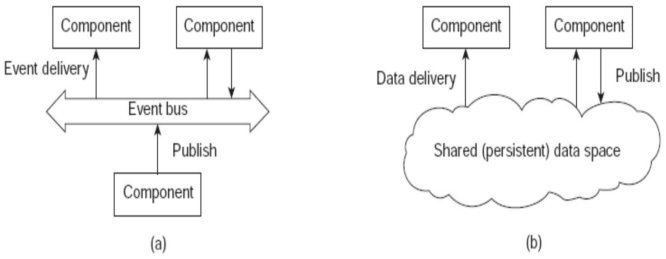
- Publish-subscribe
- Broadcast
- Point-to-point

Publish-subscribe communication style decouples sender & receiver and facilitates asynchronous communication. Event-driven architecture (EDA) promotes the production, detection, consumption of, and reaction to events[3].

The main advantage of this architecture is that they are loosely coupled.




So for this Application we are pushing the audio files to a distributed queue (Kafka ) and a multithreaded Spring Boot Application consumes those Audio files from the queue.




*B. Selection of storage platform for provisional storage of audio file.*

We need to store the audio file in a DataBase / Storage system before a consumer picks that up and starts processing . Audio files need to be stored as BLOB data . We looked at several DataBases and found NoSql database like MongoDB performs better at storing BLOB data compared to traditional RDBMS systems[4].

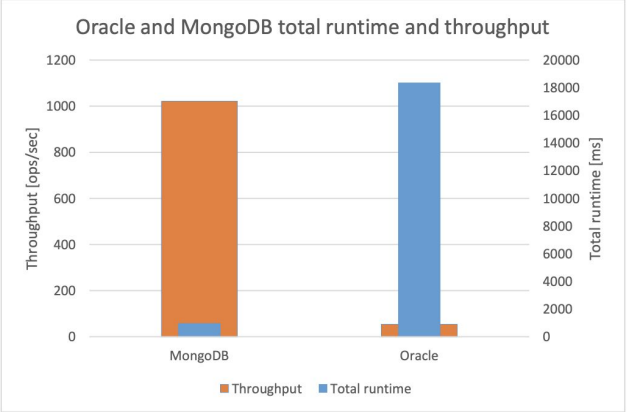



Figure 2 Workload A - Oracle and MongoDB total runtime and throughput

However, we found that storing the Audio files in MongoDB will not be a good choice , because -

a. Maximum document size supported is 16MB and large sound files can easily exceed that range .
b. If we store the audio in MongoDB , then consumer application need to constantly poll the db to check if there is a new file. If we increase the polling frequency then this new event detection is instantaneous, but it increases the system workload. So there is a tradeoff between workload and response time.

So , we **decided to use Kafka as provisional storage platform** for Audio files. Kafka performs drastically faster than other point-to-point messaging platform like Activemq or Rabbitmq[5].

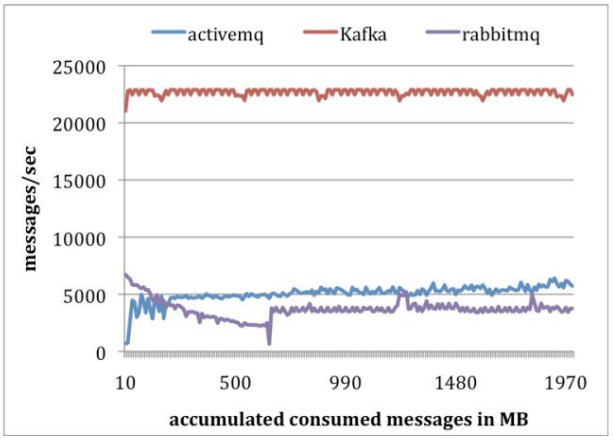



**Figure 5**. Consumer Performance

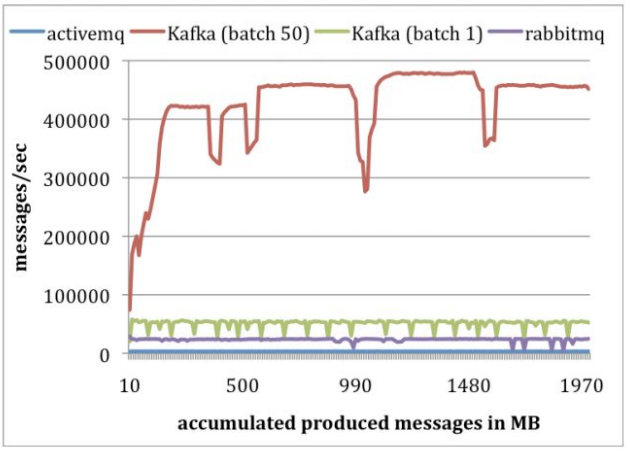



**Figure 4**. Producer Performance

*C. Storing Large files in Kafka*

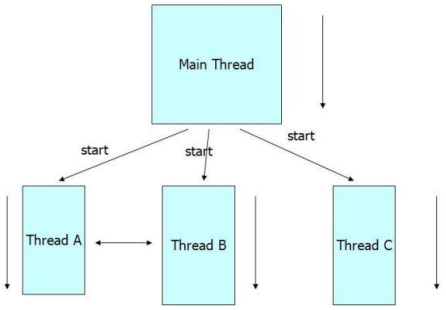
Kafka was initially designed as Distributed Messaging System for Log Processing. So default max message size ~ 1MB . A > 10 Sec 1 channel .wav file easily exceeds 1MB in size. So we need to override default configs in order to support large messages. Following properties had to be changed to support this. These values are in bytes , hence out system could easily handle ~40MB message.

```
message.max.bytes=41943040
max.request.size=41943040
replica.fetch.max.bytes=41943040
fetch.message.max.bytes=41943040
```

*D. Multithreaded kafka consumer*

Multiple producers push to kafka in a multi-partition topic, it's obvious that we also need a multi-threaded consumer to keep the whole thing robust and balanced . Previous work has been done in this topic[6] to implement a multithreaded Kafka consumer using Kafka-Python .But kafka-python is not thread safe and does not support multi-threading.

To overcome this challenge we decided to use **Spring-kafka** along with out multithreaded Spring Boot application, which is thread safe and graciously supports multithreading.




## 3. Technologies Used in System Implementation

A. **Apache Kafka** - Apache Kafka® is a distributed streaming platform.

B. **MongoDB** - MongoDB is an open-source, document database designed for ease of development and scaling.

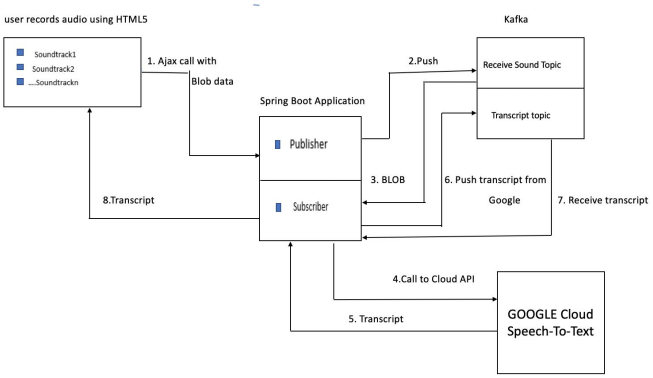
C. **Google cloud to Speech** - Powerful speech recognition cloud service that enables developers to convert audio to text by applying powerful neural network models in an easy-to-use API.

D. **Spring Boot** - Spring Boot is a Spring framework module which provides RAD (Rapid Application Development) feature to the Spring framework.

E. **Sockets** - a JavaScript library for real time web applications. It enables real time, bi-directional communication between web clients and servers.

F. **HTML5 Web Audio API** - The Web Audio API provides a powerful and versatile system for controlling audio on the Web, allowing developers to choose audio sources, add effects to audio, create audio visualizations, apply spatial effects (such as panning) and much more.

## 4. Proposed System Architecture and Design




A. We are using HTML5 MediaStream Recording API to record the audio inside a browser environment

```
navigator.mediaDevices.getUserMedia(constraints).then(function(stream) {
    /* assign to gumStream for later use */
    gumStream = stream;

    /* use the stream */
    input = audioContext.createMediaStreamSource(stream);

    /*
    Create the Recorder object and configure to record mono sound (1 channel)
    Recording 2 channels  will double the file size
    */
    rec = new Recorder(input,{numChannels:1})

    //start the recording process
    rec.record()

}).catch(function(err) {
    //enable the record button if getUserMedia() fails
    recordButton.disabled = false;
    stopButton.disabled = true;
});
```

B. Posting the Audio as BLOB data using Ajax to a Spring Boot application running on Tomcat Server

```
upload.addEventListener("click", function () {
    var xhr = new XMLHttpRequest();
    xhr.onload = function (e) {
        if (this.readyState === 4) {
            console.log("Server returned: ", e.target.responseText);
        }
    };
    var fd = new FormData();
    fd.append("audio_data", blob, filename);
    xhr.open("POST", "/api/savesound", true);
    xhr.send(fd);
});
```

C. A Spring controller receives the Audio data and pushes it to Kafka topic recieved_sound

```
@RequestMapping("/api/savesound")
public void sendMessage(@RequestParam MultipartFile  audio_data) throws IOException{
    byte[] bytes = audio_data.getBytes();
    kafkaBlobTemplate.send("recieved_sound", bytes);
}
```

D. A kafka Consumer receives an event that new audio is available and it consumes the audio and stream that audio to Google Speech-To-Text api. And when transcript is available pushes it to topic 'transcriptToClient'.

```
@KafkaListener(topics = "recieved_sound")
public void processSound(byte[] sounddata) {
    String transcript = cloudService.transcript(sounddata);
    logger.info(transcript);
    kafkaTemplate.send("transcriptToClient", transcript);
}
```

E. A service method calls to google Speech-To-Text api to get the transcript from google.

```
@Service
public class GetText {
    public String transcript(byte[] sound) {
        String finalresult = "";
        try (SpeechClient speechClient = SpeechClient.create()){
            ByteString audioBytes = ByteString.copyFrom(sound);

            // Builds the sync recognize request
            RecognitionConfig config = RecognitionConfig.newBuilder()
                .setLanguageCode("en-US")
                .build();
            RecognitionAudio audio = RecognitionAudio.newBuilder()
                .setContent(audioBytes)
                .build();
            // Performs speech recognition on the audio file
            RecognizeResponse response = speechClient.recognize(config, audio);
            List<SpeechRecognitionResult> results = response.getResultsList();

            for (SpeechRecognitionResult result : results) {
              // There can be several alternative transcripts for a given chunk of speech. Just use the
              // first (most likely) one here.
              SpeechRecognitionAlternative alternative = result.getAlternativesList().get(0);
              finalresult += alternative.getTranscript();
              //logger.info("Received Messasge in group - group-id: " + finalresult);
            }
          }catch(IOException e){
            finalresult =  e.getMessage();
        }
        return finalresult;
    }
}
```

F. When the transcript is available in 'transcriptToClient' topic a consumer picks it up and **stores it in MongoDB** for future reference. it also **pushes the transcript to a open socket** connection initiated by client.

```
@KafkaListener(topics = "transcriptToClient", groupId = "group-id
public void listen(String message) throws Exception{
    logger.info("Received Messasge: " + message);
    //Saves transcript in mongo
    repository.save(new Transcript(""+message.hashCode(), messag
    webSocket.convertAndSend("/topic/backToClient", message);
}
```

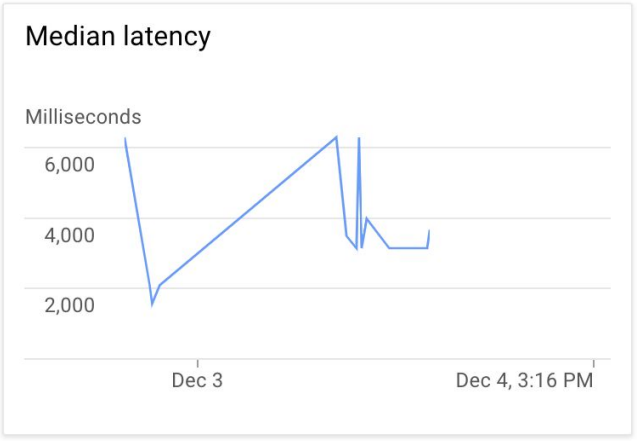
G. Client who is listening on an open socket receives the transcript and displays it in the browser

```
function connect() {
    var socket = new SockJS('/gs-guide-websocket');
    stompClient = Stomp.over(socket);
    stompClient.connect({}, function (frame) {
        setConnected(true);
        console.log('Connected: ' + frame);
        stompClient.subscribe('/topic/backToClient', function (transcript) {
            showTranscript(transcript.body);
        });
    });
}
```

## 5. Results and their analysis.

*A.Comparison of cloud services to generate text from speech*

We tried out Google Cloud Speech-to-Text, Amazon Transcribe, IBM watson Speech to Text for our experiment . We found Google Cloud Speech-to-Text performs the best compared to other too . Error rate and latency time is both significantly lower for google Speech-to-Text. Avg. **Latency time is 6sec for < 1MB** files.




*B. System load testing*

We tested our system with multiple parallel client request and a dual partition kafka topic with 2 brokers . And received following metrics in a macOs Mojave (MacBook Pro 2017 , 2.3 GHz Intel Core i5 , 8 GB 2133 MHz LPDDR3 )

Producer

```
start.time                : 2018-12-03 21:38:28:094
end.time                  : 2018-02-03 21:38:28:449
compression               : 0
message.size              : 100
batch.size                : 200
total.data.sent.in.MB     : 0.01
MB.sec                    : 0.0269
total.data.sent.in.nMsg   : 100
nMsg.sec                  : 281.6901
```

Consumer

```
start.time            : 2018-12-04 11:29:41:806
end.time              : 2018-12-04 11:29:46:854
fetch.size            : 1048576
data.consumed.in.MB   : 0.0954
MB.sec                : 1.9869
data.consumed.in.nMs  : 1001
nMsg.sec              : 20854.1667
```

*C. Audio Encodings*

We recorded mono sound (1 channel ) and noticed recording 2 channels double the file size. We used FLAC encoding , LINEAR16 and MULAW encoding separately with .wav file format . FLAC and LINEAR16 performs best with almost error free output when used for Text generation .

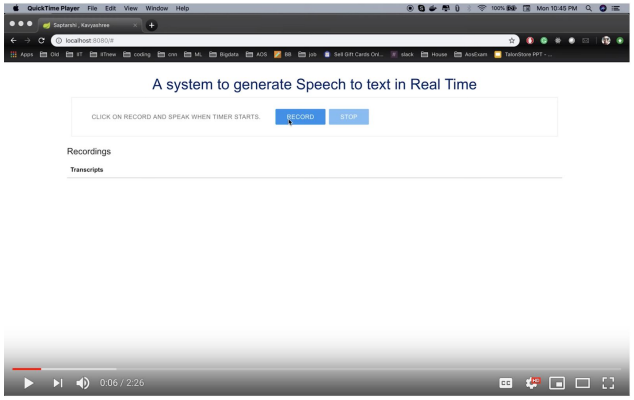
FLAC and LINEAR16 are lossless encoding compared to MULAW .

MULAW is an 8 bit PCM encoding, where the sample's amplitude is modulated logarithmically rather than linearly. As a result, uLaw reduces the effective dynamic range of the audio thus compressed.

So a higher error rate was expected for MULAW.

## 6. Demo Link.

https://www.youtube.com/watch?v=4ez2eoAz8P8&t=6s




## 7. Source Code.

https://github.com/sap9433/SpeechToText

## References


[1] Yadav, Devender. "Save Large Files in MongoDB Using Kundera - DZone Database." Dzone.com. July 24, 2016. Accessed December 01, 2018.https://dzone.com/articles/save-large-files-in-mongodb-using-kundera.

[2] Ho, Ricky. "Scalable System Design Patterns - DZone Database." Dzone.com. October 18, 2010. Accessed December 01, 2018. https://dzone.com/articles/scalable-system-design

[3] Distributed Systems Architecture . Accessed November 14, 2018. http://cse.csusb.edu/tongyu/courses/cs660/notes/distarch.php..

[4] Kamil Kolonko, “Performance comparison of the most popular relational & non-relational database management systems” IEEE http://www.diva-portal.org/smash/get/diva2:1199667/FULLTEXT02

[5] Jay Kreps,Neha Narkhede, Jun Rao, “Kafka: a Distributed Messaging System for Log Processing”. Accessed November 14, 2018. ACM 978-1-4503-0652-2/11/06 http://notes.stephenholiday.com/Kafka.pdf

[6] Guy Shilo, “Moving binary data with Kafka – a more realistic scenario”. Accessed November 14, 2018. http://www.idata.co.il/2017 /12/moving-binary-data-with-kafka-a-more-realistic-scenario/

[7] Nodar MOMTSELIDZE, “Apache Kafka - Real-time Data Processing”. Accessed November 14, 2018. https://jtst.ibsu.edu.ge/jms/index.php/jtst/article/view/80