Abstract—Virtual Reality (VR) has catapulted itself into our lives as a pleasing change in the way we interact with devices that we use in our day to day life. VR provides user a simulated environment to interact providing a more realistic experience than traditional user interface. Before diving into creating a virtual environment for an existing system it is recommended to analyze the possibility of doing so using proof of concept projects to understand the obstacles for virtualization and the limitations of the current system.

For building a application with good user experience it is required to have a robust back end system, this project is focused on building a backed system which would be a proof of concept that a virtual reality application for the on/off boxes can be created. An on/off box is a small tone generator device which has an on/off switch, a volume control, a frequency knob and a speaker as shown in fig 2. The Mana Contemporary High Concept Laboratory is considering to use the on/off box for performing to audience and can also be used in game based work. You can find more information about the project here [1]. The on/off boxes can provide a deeper musical experience to audiences, so they are also considered for live musical performances. For such crucial task we need to provide seamless experience using right hardware and equally robust software to impart the virtual experience. We want to create the same experience in virtual reality which would relieve us from the hassle of setting up the devices every time we require them.

Index Terms—Virtual Reality; Distributed Systems; Docker

I. INTRODUCTION

The project is about creating a robust back end system for the virtual reality application. We can build a simple Java application to run individually for the each component on a dedicated server which would be easy to setup and maintain. But there are disadvantages in doing so, if the server has hardware or network failures, the entire system would fail and there would be no backup system that can take over in case of major failures. This is called single point of failure. To get a good performance from the system we would need a server with pretty hefty configuration which would not be cost efficient and not scalable as well.

To overcome these shortcomings we can use virtualization software like Docker which makes use of commodity hardware, doesn’t have single point failure and is scalable. Docker also provides functionality like uniform environment, load balancing and concurrency which makes it an obvious choice.

II. BACKGROUND

A. Introduction to virtualization

The main purpose of virtualization is to effectively use the hardware resources available. A simple example of virtualization is hard disk drive partition on a computer, the disk is divided virtually into drives to make a logical separation of data for the user convenience. Initially we could only run...
one operating system on one computer then time progressed and we were able to run multiple operating system on the same computer. Now we have virtualization which utilizes hypervisor to share the the hardware resources among multiple users. The hypervisor sits on top of the hardware and can create multiple instances of the resources on demand. One of the most successful technology of modern day Cloud Computing is powered by virtualization.

B. Introduction to distributed system

Distributed system is about moving away from the centrally controlled architecture and working with a system that have multiple components that function independently and communicate with each other using message passing to achieve a common goals. For huge processes like processing big data and on-line game simulations distributed systems is the best fit. A system should adhere to certain characteristics to be a true distributed System.

- Concurrency - each individual entity in the system works in parallel to each other to accomplish a common goal.
- Independent failures - the components of the system can fail at any point in time but the failure is limited to the component itself, the rest of the system continues working.
- Scalability - the system should be able to handle the users as they grow by adding more components to the existing system which in turn increases the processing capability of the system.
- Fault tolerant - the ability of the system to handle the errors with grace so it does not lead to failure of the entire system. Errors in the system can be at multiple levels viz. Hardware, Software and Network connection.
- Transparency - the system should be rendered to the user as if it is singular system hiding the fact that there are multiple components communicating to each other.

III. Analysis

The on|off boxes created by Eastman school of music are simple tone generators, for creating a virtual version on these boxes we need to record the attributes of the boxes and store them to be recreated in the VR application. The on|off boxes have controls like volume knob, frequency knob and on-off switch, further to track the location of the physical boxes we need a location tracking device that can be placed inside the box and be able to communicate with the device to get its current location and other attributes of the box.

The on|off boxes can be used outside in a field for experimentation or can be used inside an enclosed structure, this makes it challenging to get the location as the widely used GPS devices which otherwise provide accurate current location wont work very well inside a building (enclosed structure). So it is necessary to use a location tracking device that will provide location in reference to the other boxes, for example if the boxes are set in a room we want to visualize where the boxes are as oppose to where the room itself is.

IV. Solution

A. Introduction to docker

Docker is operating system virtualization software whose main aim is to reduce the number of resources required for virtualization. Docker uses cgroup namespaces of Linux kernels and overlay file system to build Docker is different from the traditional virtualization software like VMware, Vagrant and Kubernetes. The traditional virtualization softwares have stack as shown in fig 3 where you have Hypervisor on top of host operating system and the hardware and then multiple operating systems with their independent kernels are interfacing with the
hypervisor for resource allocation, IO and other system level functionalities. This is a good model for running independent environments but the multiple guest operating systems end up taking a lot of space and resources.

Docker on other hand as shown in the fig 4 uses container for virtual environments instead of guest operating systems. Containers are lightweight and independent executable images which run on the same host operating system by sharing the resources of the host machine. Since containers require fewer resources we can deploy more of them on single computer which in turn saves a lot of cost in hardware.

B. Hardware setup

For location tracking we decided to go with DecaWave 1001M which has two components to it tags and anchors. Anchors are the devices providing the static point of reference to the tags, the anchors are placed along the walls of the room and the tags inside the on|off boxes. The tags communicate with anchors to get the physical distance between the tag and anchor in meters. One by one the tag gets its distances from all the anchors, and using triangulation algorithm it determines the relative location by providing X, Y and Z coordinate values. A minimum of fours tags are required to get the correct triangulation data.

The DecaWave chip interfaces with raspberry pie, which records the current status of the volume knob, the frequency knob and the on-off switch along with the current location coordinates. This data is then sent to the server.

This is proof of concept project so we have four anchors and only one tag. The actual system is would have around 25 on|off boxes and the same number of tags. This is where my implementation started for building a simulation module that would simulate 25 on|off boxes. The simulation process would synthesize data similar to that of an actual box would produce data viz. volume, frequency, on-off switch status and the location coordinates of the box.

C. System Architecture

Fig 5 shows the flow of the entire system The project work is focused on building a server side system using the distributed system algorithms and technologies to create a high performance backend for capturing the data from on|off boxes and visualizing the data to track the boxes. The data from on|off boxes will serve as an system input. The data will be captured by a serve and saved to the database which in this case I had decided to go with NoSQL document based database - MongoDB. Further the data will be sanitized incase we have outlier values.

Since we have multiple components in the system that run asynchronously and independent to each other but the output is dependent on the flow of the processes it is a good idea to have a monitoring system that can raise flags in case of failure and restart the system to its initial state. Once we have the location data available for tags, the visualization process will use processing language and toolkit to visualize the anchors and tags.

V. System Synthesis

The implementation started with deciding on the hardware to be used for the location tracking of the boxes and protocol to interfacing with the sever. How the server would collect data from all the boxes and visualize it.

The system is divided into 3 parts, the simulation module, the server side collection of the data from the simulation module and the visualization module.

A. Phase 1 - on|off box Simulation

The simulation system is build to emulate a physical on|off box. Similar to the on|off box the simulation system will generate data for on-off switch, volume, frequency and triangulation data (x, y and z coordinates) by synthesizing it with a particular range.

- Frequency: 1 - 100
- Volume: 1 - 10
- Triangulation data: 0 - 300 meters, since the range of decaWave tag is 300 meters.
- On-Off switch: is binary o or 1

The implementation of this module is done in Java 8 using threads which run every 2 secs to generate data. We have to simulate 25 instances of the on|off box, this is where the docker comes in.

Every server holds five containers each, 25 instances spanned across 5 servers so the load is balanced between them. Every container runs the on|off box simulation program i.e every container represents a physical box.

B. Phase 2 Receiving server

The server can handle incoming data from 25 containers simultaneously. The server program is written in Java using threads and User Datagram Protocol. The single instance of the program can handle more than 50 connections at a given
time. The receiving server runs on Docker containers that has a Mongo database instance running as well.

C. Docker Compose

- Running the system on containers provides a uniform platform which ensures that the program runs on any machine regardless of the hardware configuration as the environment is uniform across the containers.
- Considering we are running 30 docker containers, it is tedious to run all the docker containers individually. So we can use Docker compose to run the containers using a single script.
- Using compose we can set the application services for the containers running by creating yaml files.

1) Docker Swarm:

- Running a high number of docker containers is tedious and difficult to manage, if any of the containers go down due to network failures or hardware failure it can affect system performance.
- To deal with the Docker containers clustering and scheduling Swarm is an effective tool. Swarm can orchestrate redundancy between the docker containers, run a fail safe mode which spawns a new docker container when there is a failure. Swarm also help to add new container or remove containers based on computational needs.
- Swarm works as a layer between the Hypervisor and the containers so it can also perform load balancing, for the on/off box simulation the number of containers to run on a single machine are restricted to 5. Consider running the on/off box simulation on single computer and we want to add more containers but there are no resources available,
we can add another machine with Docker installed on it and it will join the cluster and Swarm will move the new instances the new machine thus increasing the operational capabilities for the system.

2) **Mongo Database Sharding**: There are multiple containers receiving the data and saving it to its own copy of the MongoDB each container data has to be replicated to the other containers as well for data consistency. This is where the sharding in MongoDB is the most effective. We already have a cluster available from Docker Swarm which can be leveraged for storing chunks of databases on the individual containers. The data on each container is a shard of the entire database that is spread across the containers. There are three components to the MongoDB sharding mechanism.

1) **Mongo routers** - the router redirects the query to appropriate shard and then return the result back.

2) **Mongo shards** - each shard is used to store data and has a replica set on other machine for data consistency and high availability.

3) **Mongo config** - stores the cluster metadata about the shards location and replication sets. The router uses the config to redirect the query request to the shards.

### D. Phase 3 - Visualization using Processing

Processing is a programming language and an IDE for creating visual designs. Processing has support for all standard data structures in Java and JSON. The location data that is recorded in MongoDB for on-off boxes is used to visualize the locations of the boxes with respect to the anchors in 2 dimensions. As we have used four anchors, they are placed in the four corners of the canvas for intents and purposes of this project.

There are x, y and z coordinates available for each box. We are not going to use the z coordinate as elevation doesn't play any role in localizing the box in a room in two dimensions. Since processing is built on top of Java programming language it is possible to write a pure Java code for processing. To maintain the consistency of the programming language used across the system I decided to import the processing jar file in the project and use Java for visualization purposes.

As you can see fig 8 for initial implementation had only 10 tags and two anchors for simplicity.

![Fig. 7. Tag simulation Docker architecture](image)

![Fig. 8. Initial visualization for on—off boxes](image)

### VI. Results

- The system was tested with 25 instances of on-off box containers running on the servers sending the data to the receiving servers running on different machine.
Each of the containers have chunk of data stored in MongoDB.
- The system is stable running 30 containers and there is no latency in the location change of the boxes.
- The range of the DecaWave chip is limited to 300 meters, for visualization purposes we have scaled the module to represent the distance in millimeters. Which gives us 300 x 300 millimeter area to display 25 tags. The visualization appears to be very cluttered and not all tags are visible due to overlapping.
- For a helpful and clutter free visualization I have increased the range of the DecaWave anchors in the simulation and hence we can see in fig 9 that the coordinates of some boxes are greater than 300.

A. Result verification

The resulting visualization was tested in two different ways
- Visual verification
  - Assigning specific locations for the boxes beforehand in the database then visualizing them and check if the locations are as expected.
  - For this test I had only used 10 boxes so it was easy to determine the location and the visualization was uncluttered and manageable.
- Verification using Processing
  - There is a Java program written for verification of the results called Verification.java. This program first reads the data from Mongo database for all the boxes. We have data for X,Y and Z coordinated for anchors 1,2,3 and 4.

  - Next we calculate the distance between the position of the on/off box which is placed using the anchor 1 coordinates and the remaining of the anchors. This data is written back to the database so we have the distances as reference which is calculated programmatically.

  - Then we render the on/off boxes but additionally draw a line between the on/off box and the anchor for whom we want to verify the data for.

  - The we can compare the original distance we had calculate and written in the database with the length of the line drawn.

  - The original distances are present in the database and the length of the line is present on the box itself for all the anchors.

  - We can compare the values for every tag a see that the values are correct for the coordinates were the on/off box is placed. This is more accurate method to verify the visualization with the simulation data.

VII. CONCLUSION

- Using DecaWave chip-set and Raspberry pie we can enable the on/off boxes to provide location and other feature data.
- The server side system is fault tolerance, consistent, fast and reliable as it is virtualized using Docker along with Docker Compose and Swarm.
- The visualization, built using Processing language and toolkit provides a substantial proof that we can build a Virtual Reality application which would take data input
from the on/off box simulation module and provide users a wholesome experience of the tone generator boxes.

VIII. FUTURE WORK

- The system is built to a point where it possible to create a virtual reality application using the data from the server side system.
- Currently we are using only x and y coordinates from the x, y and z coordinates provided by the anchor. Since we are using 2 dimensional visualization. The z coordinate values which represent the elevation can be leveraged in the VR application for placing the boxes at different heights.

IX. APPENDIX

A. Common settings

- Install latest version of Mongo database 3.6.
- Install Robomongo or equivalent GUI tool to connect to Mongodb.
- Create a database called 'OnOffBox' and collection called 'BoxData'.
- Install virtual box.
- Install Docker.

The system can be executed in two modes.

B. System execution

1) Local single machine run mode:

- Running the system in this mode is pretty straight forward. You have to run two scripts which would start the on/off box simulation module and the receiving server.
- To start the on/off box simulation run the BoxSimulation.sh script using command.

\[
\text{bash BoxSimulation.sh}
\]

The script will pop up 15 windows each representing one on/off box.
- Once the simulation is running it will emit data to localhost port 40000 using UDP. To receive this data run the script to start the receiving server using following command.

\[
\text{bash ReceivingServer.sh}
\]

- Once the receiving server starts receiving data it write the data to MongoDB database called 'OnOffBox'. The database has a collection called BoxData which stores data for all the 25 on/off boxes.
- Finally run the script for visualization which will graphically render the on/off boxes.

\[
\text{bash Visualization.sh}
\]

2) Docker mode:

- To run the system over the Docker the ideal system setup would be to use 5 machines to run the on/off box simulation, two machines to run the receiving server, the visualization process can be run over the same machines as the receiving server.
- We are using Docker in Swarm mode for orchestrating the docker containers. Following is the command to start the Docker Swarm.

\[
\text{docker swarm init --advertise-addr }
\langle\text{any-ip-address-you-like}\rangle
\]

example

\[
\text{docker swarm init --advertise-addr }\langle\text{192.168.101.134}\rangle
\]

- This will create single node docker swarm. The ip address in the above command is used by the other nodes to connect with swarm.
- Use following command to check how many nodes are running in docker currently.

\[
\text{docker node ls}
\]

- With the fault tolerance perspective if you have N+1 node manager the swarm keeps on running until N/2 managers go down.
- Adding nodes to the Swarm
  - To add nodes to swarm we need to get token for the swarm. Use following command to get the token

\[
\text{docker swarm join-token manager}
\]
- The to join the swarm as a manager node use following command

```bash
docker swarm join --token <token> \<ipaddress—we-provided-earlier>
```

- To add a worker node, we first need to get the token for the worker node

```bash
docker swarm join-token worker
```

- Add the worker node

```bash
docker swarm join --token <token> \<ipaddress—we-provided-earlier>
```

- We need to add five worker nodes on each of the on/off box simulation servers. Two worker nodes to each receiving server machine, one for receiving server program and one for the visualization program.
- Now we have to run the Java program in the worker nodes we have created on each of the machines. There are three Docker script called Dockerfile.
  - Dockerfile for on/off box simulation has the Java command to run the program with the tag number.
  - Dockerfile for receiving server runs to the Java program and writes to the Mongodb.
  - Dockerfile for visualization runs the processing script to the read data form Mongodb and renders the boxes.
- Run the docker container on each machine using appropriate docker file

```bash
docker run -i -t -p <port to expose>:<mapping port in docker> -v ~$/docker/db/ <name of the container>
```

**ACKNOWLEDGMENT**

I would like to thank Dr. Joe Geigel for providing an opportunity to work on the project for Eastman College of Music, for providing me valuable feedback and gracefully guiding me through the challenging project work.

**REFERENCES**
