INDIAN INSTITUTE OF INFORMATION TECHNOLOGY

Dharwad
ज्ञानेन विकासः

Big Data Analytics and Ubiquitous Computing

Dr. Animesh Chaturvedi

Assistant Professor: IIIT Dharwad

Post Doctorate: King's College London & The Alan Turing Institute

PhD: IIT Indore MTech: IIITDM Jabalpur

KING'S College LONDON

Indian Institute of Technology Indore
भारतीय प्रौद्योगिकी संस्थान इंदौर

PDPM
Indian Institute of Information Technology, Design and Manufacturing, Jabalpur

The Alan Turing Institute

Big Data Analytics

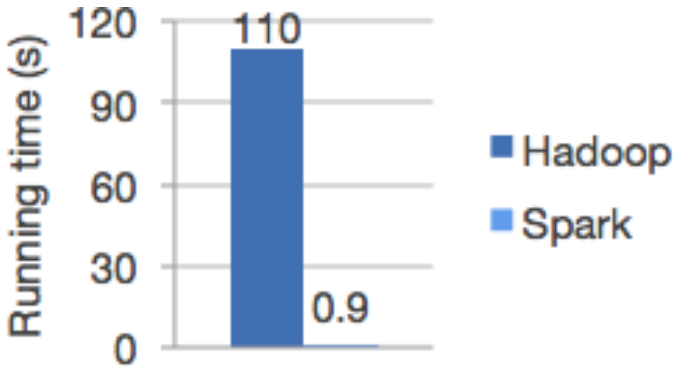
- **Big Data Analytics**
  1. **Big Data**
  2. **Spark: Big Data Analytics**
  3. **Resilient Distributed Datasets (RDD)**
  4. **Spark libraries (SQL, DataFrames, MLlib for machine learning, GraphX, and Streaming)**
  5. **PFP: Parallel FP-Growth**
- Ubiquitous Computing

Big Data

- Big data can be described by the following characteristics:
  - Volume: size large than terabytes and petabytes
  - Variety: type and nature, structured, semi-structured or unstructured
  - Velocity: speed of generation and processing to meet the demands
  - Veracity: the data quality and the data value
  - Value: Useful or not useful
- The main components and ecosystem of Big Data
  - Data Analytics: data mining, machine learning and natural language processing
  - Technologies: Business Intelligence, Cloud computing & Databases
  - Visualization: Charts, Graphs etc.

Apache Spark

- Unified analytics engine for large-scale data processing.
- Speed: Run workloads 100x faster.
- Both batch and streaming data, using Directed Acyclic Graph (DAG) scheduler, a query optimizer, and a physical execution engine.
- Ease of Use: Write applications quickly in Java, Scala, Python, R, and SQL.
- Spark offers 80+ high-level operators to build parallel apps.



https://spark.apache.org/

Spark: Unified Big Data Analytics

- New applications of Big data workloads on Unified Engine of
  - Streaming, Batch, and Interactive.
- Composability in programming libraries for big data and encourages development of interoperable libraries
- Combining the SQL, machine learning, and streaming libraries in Spark

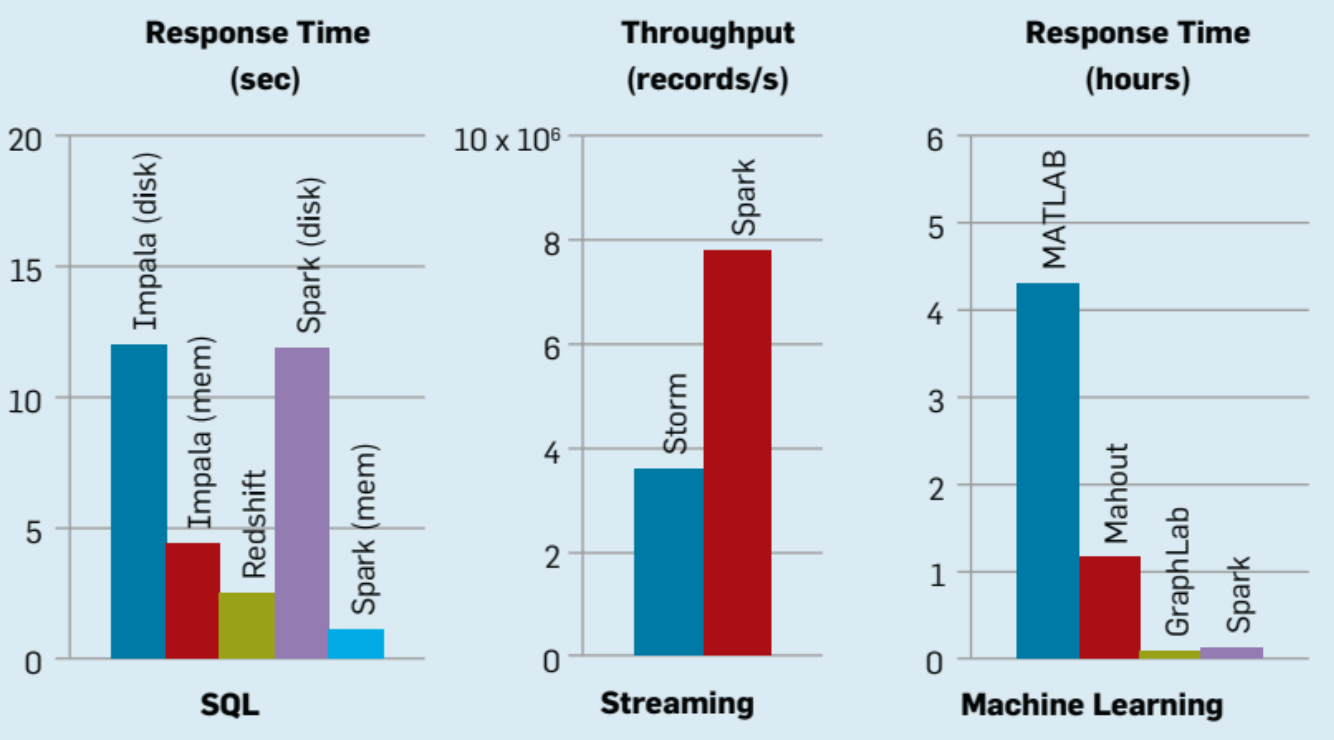
```
// Load historical data as an RDD using Spark SQL
val trainingData = sql(
   "SELECT location, language FROM old_tweets")

// Train a K-means model using MLlib
val model = new KMeans()
   .setFeaturesCol("location")
   .setPredictionCol("language")
   .fit(trainingData)
// Apply the model to new tweets in a stream
TwitterUtils.createStream(...)
      .map(tweet => model.predict(tweet.location))
```

Zaharia, Matei, et al. "Apache spark: a unified engine for big data processing." *Communications of the ACM* 59.11 (2016): 56-65.

Spark: Unified Big Data Analytics

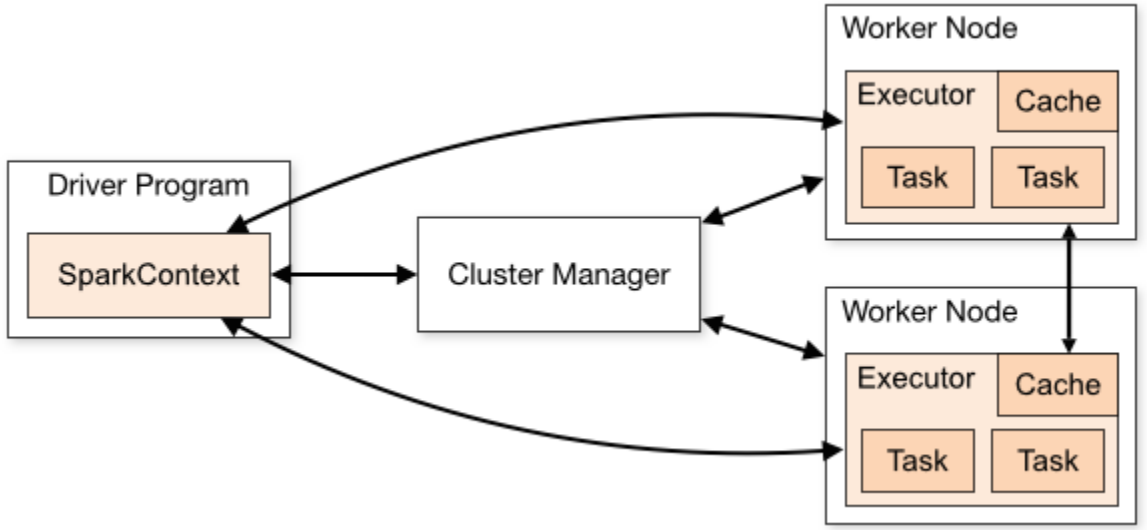
- Spark has MapReduce programming model with extended data-sharing abstraction called "Resilient Distributed Datasets," or RDDs.



Zaharia, Matei, et al. "Apache spark: a unified engine for big data processing." *Communications of the ACM* 59.11 (2016): 56-65.

Spark Architecture

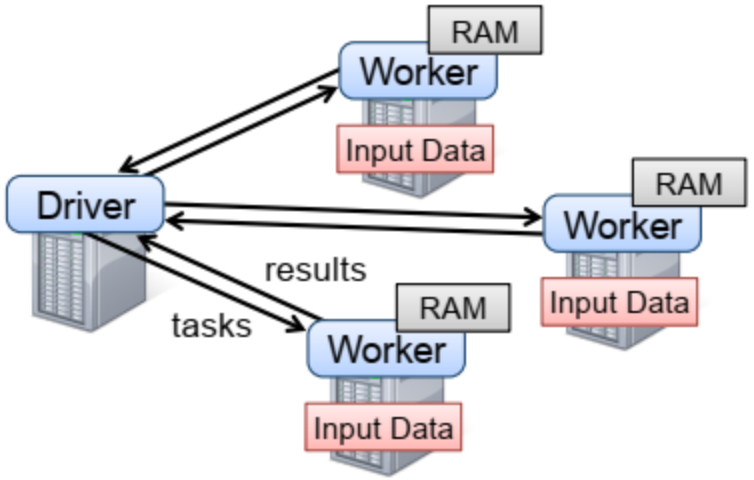
- Spark works on the popular Master-Slave architecture.
- Cluster works with a single master and multiple slaves.
- The Spark architecture depends upon two abstractions:
  - Resilient Distributed Dataset (RDD)
  - Directed Acyclic Graph (DAG)



https://spark.apache.org/docs/latest/cluster-overview.html

Resilient Distributed Datasets (RDD)

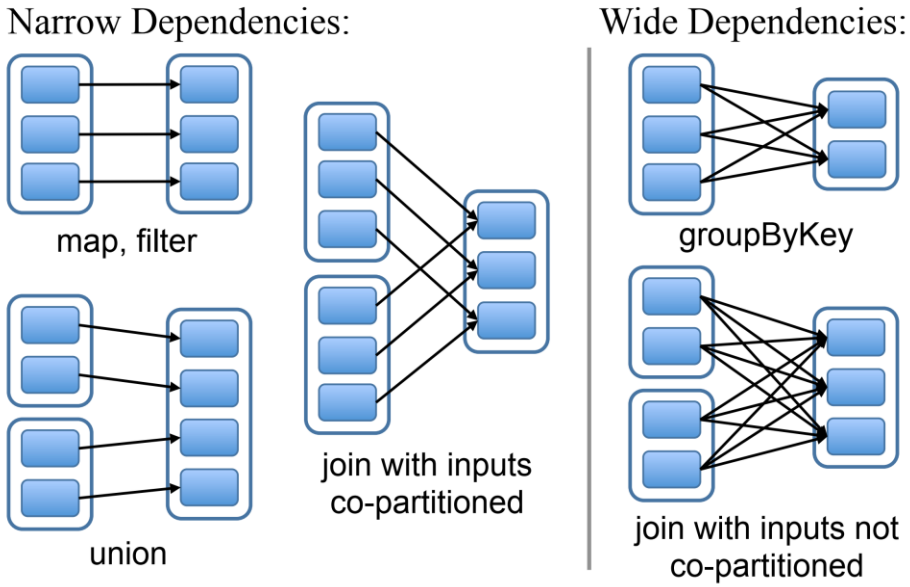
- A distributed memory abstraction: perform in-memory computations on large clusters
- Keeping data in memory can improve performance
- Spark runtime: Driver program launches multiple workers that read data blocks from a distributed file system and can persist computed RDD partitions in memory.



Zaharia, Matei, et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*. 2012.

Resilient Distributed Datasets (RDD)

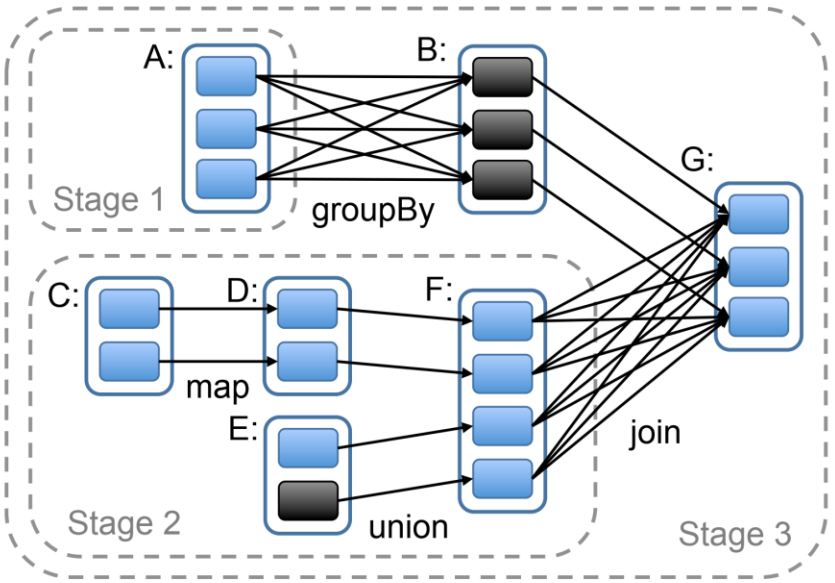
- Each box is an RDD, with partitions as shaded rectangles.
- *narrow* dependencies, where each partition of the parent RDD is used by at most one partition of the child RDD,
- *wide* dependencies, where multiple child partitions may depend on it.



Zaharia, Matei, et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*. 2012.

Resilient Distributed Datasets (RDD)

- Direct Acyclic Graph (DAG) to perform a sequence of computations
- Each node is an RDD partition,
- Run an action (*e.g.*, *count* or *save*) on an RDD, the scheduler examines that RDD's lineage graph to build a DAG of *stages* to execute.
- Each stage contains with narrow dependencies
- The boundaries of the stages are the shuffle operations required for wide dependencies.
- Scheduler computes missing partitions until it computed RDD.



Zaharia, Matei, et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*. 2012.

Apache Spark

- Combine SQL, streaming, and complex analytics. Spark libraries
  - SQL and DataFrames,
  - MLlib for machine learning,
  - GraphX, and
  - Spark Streaming.
- Spark runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud.
- Run Spark using its standalone cluster mode, on EC2, on Hadoop YARN, on Mesos, or on Kubernetes.
- Access data in HDFS, Alluxio, Apache Cassandra, Apache HBase, Apache Hive, and hundreds of other data sources.

https://spark.apache.org/

Spark Code Example

- Word Count

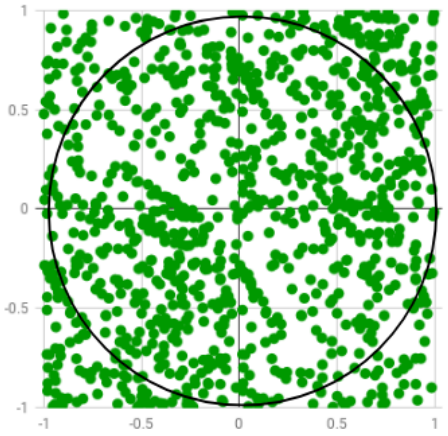
```
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts = textFile
    .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
    .mapToPair(word -> new Tuple2<>(word, 1))
    .reduceByKey((a, b) -> a + b);
counts.saveAsTextFile("hdfs://...");
```

- Pi Estimation

```
List<Integer> l = new ArrayList<>(NUM_SAMPLES);
for (int i = 0; i < NUM_SAMPLES; i++) {
  l.add(i);
}

long count = sc.parallelize(l).filter(i -> {
  double x = Math.random();
  double y = Math.random();
  return x*x + y*y < 1;
}).count();
System.out.println("Pi is roughly " + 4.0 * count / NUM_SAMPLES);
```

$$\text{Pi} = 4 \times \frac{number\ of\ random\ point\ inside\ the\ circle}{number\ of\ random\ point\ inside\ the\ square}$$



Spark SQL

- Working with structured data.
- Integrated: SQL queries with Spark programs.

```
results = spark.sql("SELECT * FROM people")
names = results.map(lambda p: p.name)
```

Apply functions to results of SQL queries.

- Uniform Data Access: Connect to data sources, including Hive, Avro, Parquet, ORC, JSON, and JDBC.

```
spark.read.json("s3n://…").registerTempTable("json")
results = spark.sql("""SELECT * FROM people JOIN json …""")
```

- Query and join different data sources
- Hive Integration Spark HiveQL
- Standard Connectivity: Connect through JDBC or ODBC.
- Business intelligence tools to query big data.

https://spark.apache.org/sql/

DataFrame API Examples

- Collection of data organized into named columns
- Use DataFrame API to perform various relational operations
- Automatically optimized by Spark's built-in optimizer

```
// Creates a DataFrame having a single column named "line"
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaRDD<Row> rowRDD = textFile.map(RowFactory::create);
List<StructField> fields = Arrays.asList(
  DataTypes.createStructField("line", DataTypes.StringType, true));
StructType schema = DataTypes.createStructType(fields);
DataFrame df = sqlContext.createDataFrame(rowRDD, schema);

DataFrame errors = df.filter(col("line").like("%ERROR%"));
// Counts all the errors
errors.count();
// Counts errors mentioning MySQL
errors.filter(col("line").like("%MySQL%")).count();
// Fetches the MySQL errors as an array of strings
errors.filter(col("line").like("%MySQL%")).collect();
```

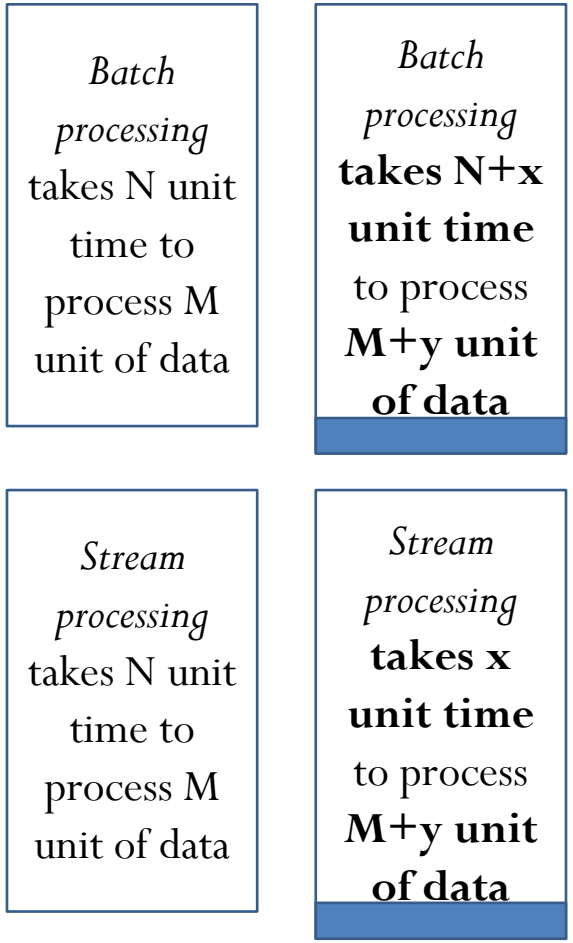
Spark Streaming

- Build scalable fault-tolerant streaming applications.
- Write streaming jobs -- Same Way -- Write batch jobs
  - Counting tweets on a sliding window

```
TwitterUtils.createStream(…)
.filter(_.getText.contains("Spark"))
.countByWindow(Seconds(5))
```

- Reuse the same code for batch processing
  - Find words with higher frequency than historic data:

```
stream.join(historicCounts).filter {
  case (word, (curCount, oldCount)) =>
    curCount > oldCount
}
```



https://spark.apache.org/streaming/

Spark GraphX

- Spark's API for graphs and graph-parallel computation

```
graph = Graph(vertices, edges)
messages = spark.textFile("hdfs://…")
graph2 = graph.joinVertices(messages) {
  (id, vertex, msg) => …
}
```

- Fast Speed for graph algorithms
- GraphX graph algorithms
  - PageRank
  - Connected components
  - Label propagation
  - SVD++
  - Strongly connected components
  - Triangle count

https://spark.apache.org/graphx/

Spark MLlib

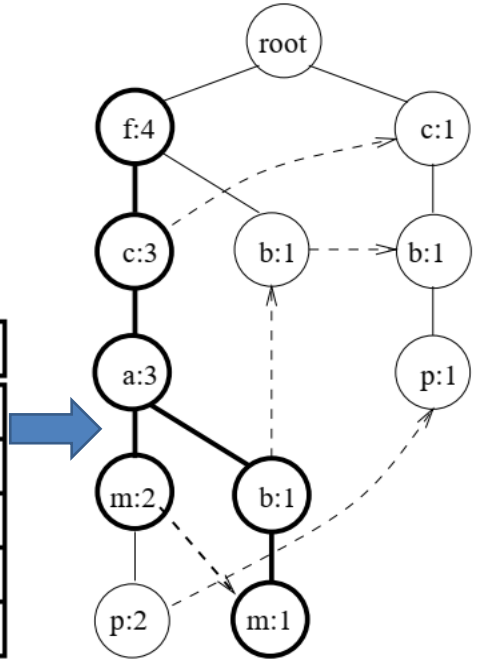
- Spark's scalable machine learning library
- Spark MLlib algorithms
  - Classification: logistic regression, naive Bayes,…
  - Regression: generalized linear regression, survival regression,…
  - Decision trees, Random forests, and Gradient-boosted trees
  - Recommendation: Alternating Least Squares (ALS)
  - Clustering: K-means, Gaussian mixtures (GMMs),…
  - Topic modeling: Latent Dirichlet Allocation (LDA)
  - **Frequent itemsets, Association rules, and Sequential pattern mining**

https://spark.apache.org/graphx/

FP-Growth for recommendation

- "FP" stands for Frequent Pattern in a Dataset of transactions
  1. calculate item frequencies and identify frequent items,
  2. a suffix tree (FP-tree) structure to encode transactions, and
  3. frequent itemsets can be extracted from the FP-tree.
- Input: Transaction database
- Intermediate Output: FP-Tree
- Output: {f, c, a → a, m p}, {f, c, a → b, m}

TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p



Han Jiawei, Jian Pei, and Yiwen Yin. "Mining frequent patterns without candidate generation." *ACM SIGMOD Record* 29.2 (2000): 1-12.

PFP: Parallel FP-Growth

- In Spark ML-Library (MLLib), a parallel version of FP-growth called PFP: Parallel FP-Growth
- PFP distributes the work of growing FP-trees based on the suffixes of transactions.
- More scalable than a single-machine implementation.
- PFP partitions computation, where each machine executes an independent group of mining tasks

Li, Haoyuan, et al. "PFP: Parallel FP-Growth for query recommendation." *Proceedings of the 2008 ACM Conference on Recommender systems*. 2008.

PFP: Parallel FP-Growth

Example of MapReduce FP-Growth: Five transactions composed of lower-case alphabets representing items

Map inputs (transactions) key="": value	Sorted transactions (with infrequent items eliminated)	Map outputs (conditional transactions) key: value
f a c d g i m p	f c a m p	p: f c a m m: f c a a: f c c: f
a b c f l m o	f c a b m	m: f c a b b: f c a a: f c c: f
b f h j o	f b	b: f
b c k s p	c b p	p: c b b: c
a f c e l p m n	f c a m p	p: f c a m m: f c a a: f c c: f

Reduce inputs (conditional databases) key: value		Conditional FP–trees
p:	{ f c a m / f c a m / c b }	{(c:3)} | p
m:	{ f c a / f c a / f c a b }	{ (f:3, c:3, a:3) } | m
b:	{ f c a / f / c }	{} | b
a:	{ f c / f c / f c }	{ (f:3, c:3) } | a
c:	{ f / f / f }	{ (f:3) } | c

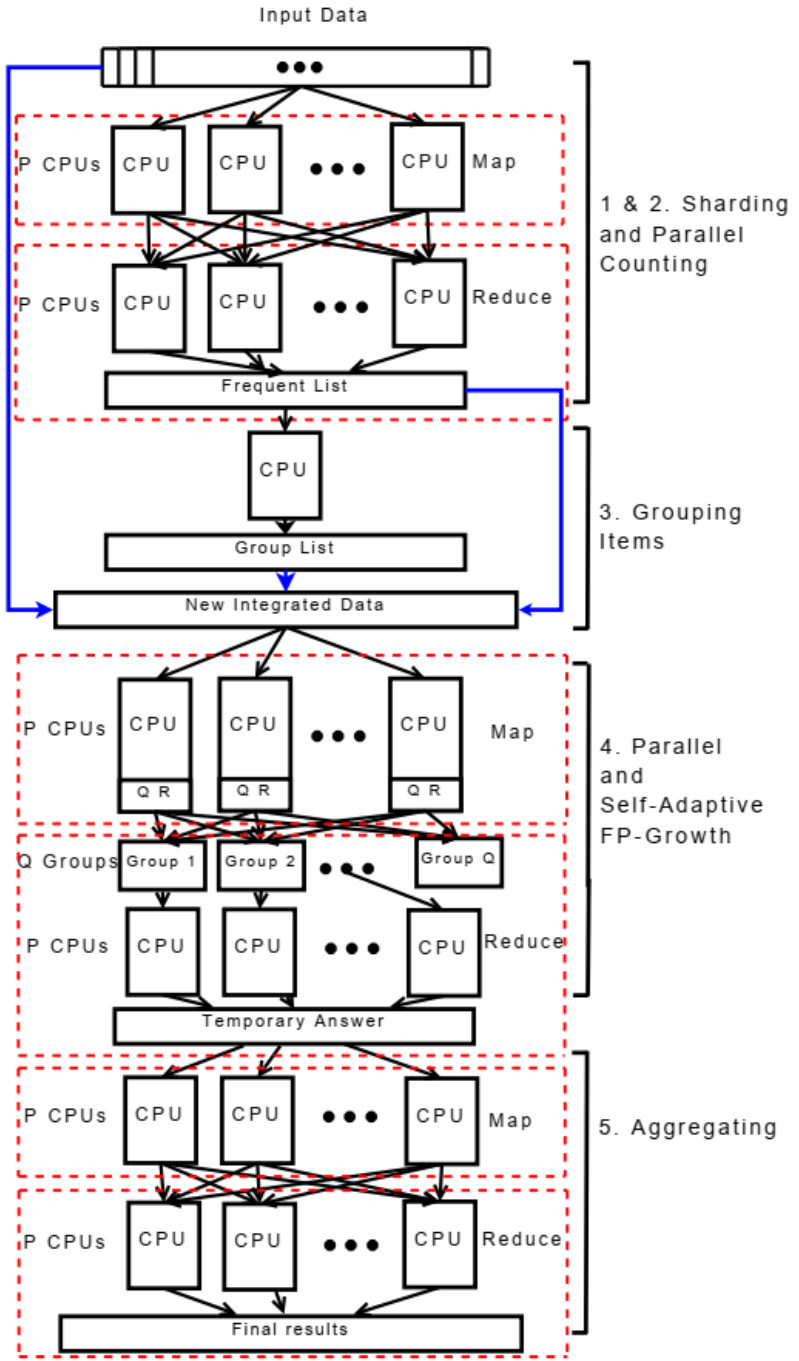
Li, Haoyuan, et al. "PFP: Parallel FP-Growth for query recommendation." *Proceedings of the 2008 ACM Conference on Recommender systems*. 2008.

- **Sharding:** Divide DB into successive parts and storing the parts (as a Shard) on P different computers.
- **Parallel Counting:** MapReduce counts the support of all items that appear in DB. Each mapper inputs one shard of DB. The result is stored in F-list.
- **Grouping Items:** Dividing all the items on F-List into Q groups of a list (G-list).
- **Parallel FP-Growth:** A MapReduce

  - Mapper: Each mapper uses a Shard. It reads a transaction in the G-list and outputs one or more key-value pairs, where each key is a *group-id* and value is a **group-dependent transaction**.

  - For each *group-id*, the MapReduce groups all group-dependent transactions into a shard.

  - Reducer: Each reducer processes one or more group-dependent Shard. For each shard, a reducer builds a local FP-Tree and discover patterns.
- **Aggregating:** Aggregate the results generated as final result.



Li, Haoyuan, et al. "PFP: Parallel FP-Growth for query recommendation." *ACM Conf. on Recommender systems*. 2008.

PFP: Parallel FP-Growth

- FP-Growth implementation takes the following (hyper-)parameters
  - minSupport: the minimum support for an itemset to be identified as frequent e.g., if an item appears 3 out of 5 transactions, it has a support of 3/5=0.6.
  - minConfidence: minimum confidence for generating Association Rule e.g., if in the transactions itemset X appears 4 times, X andY co-occur only 2 times, the confidence for the rule X =>Y is then 2/4 = 0.5.
  - numPartitions: the number of partitions used to distribute the work.
- FP-Growth model provides:
  - freqItemsets: frequent itemsets in the format of DataFrame("items"[Array], "freq"[Long])
  - associationRules: association rules generated with confidence above minConfidence, in the format of DataFrame("antecedent"[Array], "consequent"[Array], "confidence"[Double]).

PFP: Parallel FP-Growth

```
import java.util.Arrays;
import java.util.List;

import org.apache.spark.ml.fpm.FPGrowth;
import org.apache.spark.ml.fpm.FPGrowthModel;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.RowFactory;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.types.*;

List<Row> data = Arrays.asList(
  RowFactory.create(Arrays.asList("1 2 5".split(" "))),
  RowFactory.create(Arrays.asList("1 2 3 5".split(" "))),
  RowFactory.create(Arrays.asList("1 2".split(" ")))
);
StructType schema = new StructType(new StructField[]{ new StructField(
  "items", new ArrayType(DataTypes.StringType, true), false, Metadata.empty())
});
Dataset<Row> itemsDF = spark.createDataFrame(data, schema);

FPGrowthModel model = new FPGrowth()
  .setItemsCol("items")
  .setMinSupport(0.5)
  .setMinConfidence(0.6)
  .fit(itemsDF);

// Display frequent itemsets.
model.freqItemsets().show();

// Display generated association rules.
model.associationRules().show();

// transform examines the input items against all the association rules and summarize the
// consequents as prediction
model.transform(itemsDF).show();
```

Ubiquitous Computing

- Big Data Analytics (Apache Spark (SQL, Mllib, GraphX))
- **Ubiquitous Computing**
  1. **Edge Computing**
  2. **Cloudlet**
  3. **Fog computing**
  4. **Internet of Things (IoT)**
  5. **Virtualization**
  6. **Virtual Conferencing**
  7. **Virtual Events (2D, 3D, and Hybrid)**

Ubiquitous computing

- Mark Weiser: Three basic ubiquitous computing devices:
  - Tabs: a wearable device that is approx in centimeters
  - Pads: a hand-held device that is approximately a decimeter in size
  - Boards: an interactive larger display device that is approximately a meter in size
- computing is made to appear anytime and everywhere
- any device, in any location, and in any format

Weiser, Mark. "The computer for the 21st century." *ACM SIGMOBILE mobile computing and communications review* 3.3 (1999): 3-11.
https://en.wikipedia.org/wiki/Ubiquitous_computing

Edge computing

- Distributed computing paradigm
- Computation and data storage closer to the user location
- Improve response times and save bandwidth
- Cloud computing operates on big data, whereas Edge computing operates on "instant data"
- Content Delivery Network or Content Distribution Network (CDN) (Refer to Unit 4)
- Akamai CDN (Refer to Unit 4)
- Akamai-Facebook's Photo-Serving Stack (Refer to Unit 4)

https://en.wikipedia.org/wiki/Edge_computing

Cloudlet

- First coined by Mahadev Satyanarayanan (Satya), Victor Bahl, Ramón Cáceres, and Nigel Davie
- It is a mobility-enhanced small-scale cloud datacenter that is located at the edge of the Internet.
- It work as a *data center in a box* which *brings the cloud closer.*
- Support resource-intensive and interactive mobile applications by providing powerful computing resources to mobile devices with lower latency.

https://en.wikipedia.org/wiki/Cloudlet

Fog computing

- Architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing.
- Fog computing is often erroneously called edge computing, but there are key differences.
- Fog works with the cloud, whereas edge is defined by the exclusion of cloud.
- Fog is hierarchical where edge tends to be limited to a small number of layers.
- Cloud computing deal with Big Data, whereas Fog computing deals with real-time data generated by sensors or users.

[1] IEEE Standard Association. "IEEE 1934-2018-IEEE Standard for adoption of OpenFog reference architecture for fog computing." (2018).
https://en.wikipedia.org/wiki/Fog_computing

Internet of Things (IoT)

- The network of physical objects —"things"— embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the Internet.
- Example:
  - "Smart Home" devices and appliances,
  - "Smart city" equipment and facilities,
- Real-Time Data Analytics
- *Information explosion* or *Data Deluge*
  - due to *data flood* or *information flood*
  - ever-increasing amount of electronic data exchanged per time unit
  - unmanageable amounts of data growth V/S power of data processing

https://en.wikipedia.org/wiki/Internet_of_things

Virtualization

- Virtual computer hardware platforms, storage devices, and computer network resources.
- *Hardware virtualization* or *platform virtualization* refers to the creation of a Virtual Machine (VM).
  - Full virtualization – complete simulation of the actual hardware to allow software environments, guest operating system and its apps.
  - Paravirtualization – the guest apps are executed in their own isolated domains.
- Application virtualization and Workspace virtualization: isolating individual apps from the underlying OS and other apps
- Service virtualization: emulating the behavior of API, Cloud and SOA

https://en.wikipedia.org/wiki/Virtualization

Virtualization

- Memory virtualization: aggregating RAM resources from networked systems into a single memory pool
- Virtual memory: giving an app the impression that it has contiguous working memory, isolating the underlying physical memory
- Storage virtualization: the process of completely abstracting logical storage from physical storage
- Distributed file system: any file system that allows access to files from multiple hosts sharing via a computer network
- Virtual file system: an abstraction layer on top of a more concrete file system
- Storage hypervisor: combines physical storage resources into one or more flexible pools of logical storage

https://en.wikipedia.org/wiki/Virtualization

Virtualization

- Virtual disk: emulates a disk drive such as a hard disk drive or optical disk drive
- Data virtualization: the presentation of data as an abstract layer, independent of underlying database systems, structures and storage
- Database virtualization: the decoupling of the database layer, which lies between the storage and application layers
- Network virtualization: creation of a virtualized network addressing space within or across network subnets
- Virtual private network (VPN): replaces the actual wire or other physical media in a network with an abstract layer

https://en.wikipedia.org/wiki/Virtualization

Virtual Conferencing

- Teleconference: Phone lines, Landlines or Cellular devices
- Video conference:
  - Webcam, Microphone, Speaker, Internet
  - hardware, software, devices is dedicated for this
- Web Conference: Internet and Cloud supported
  - Web 2.0
  - Well-Known: Google Meet, Skype, and Microsoft Team
  - Multi-Communications from Many sender to Many receivers
  - Webinars ("web seminars"), Webcasts (live media presentation), Podcast (audio presentation), and web meetings
  - **Virtual Events**

https://en.wikipedia.org/wiki/Internet_of_things

Virtual Events

- An online event involves people interacting in a virtual environment on the web, instead of physical meeting.
- Multi-session online events often feature webinars and webcasts.
- Aim to create similar experience as physical meeting.
- Live-streaming the event online or on-demand video.
- Issues
  - Echo of voice
  - Audio and Videography logistics
  - Network Bandwidth of conferencing server and users

https://en.wikipedia.org/wiki/Virtual_event

2D Virtual Events: Edited Collage

2D Virtual Events: Virtual Degree

2D Virtual Events: Virtual Address

3D Virtual Event: Live Video

[Virtual HLF 2020 - Heidelberg Laureate Forum]

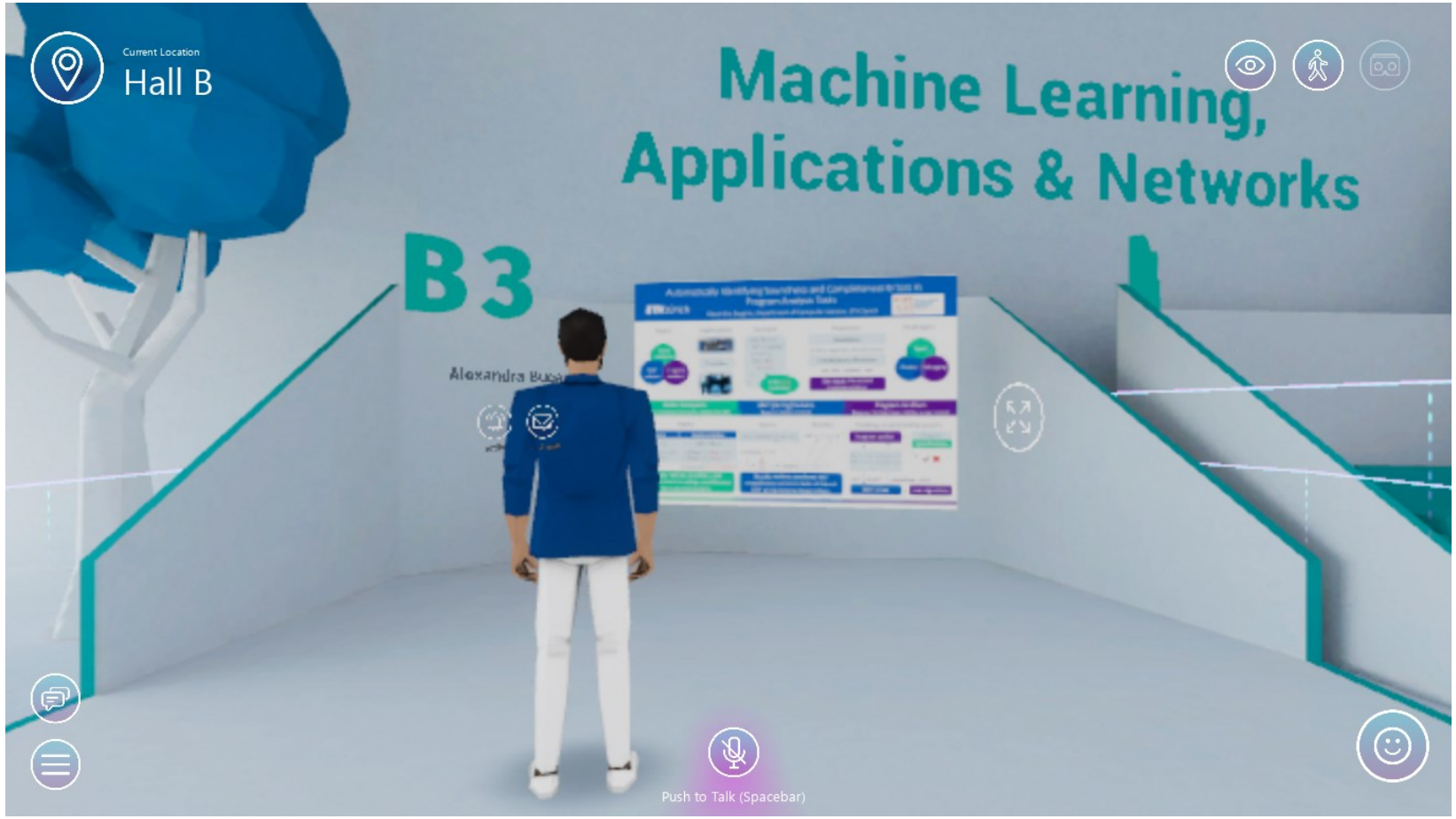
3D Virtual Event:

[Virtual HLF 2020 - Heidelberg Laureate Forum]

3D Virtual Event: Poster Sessions

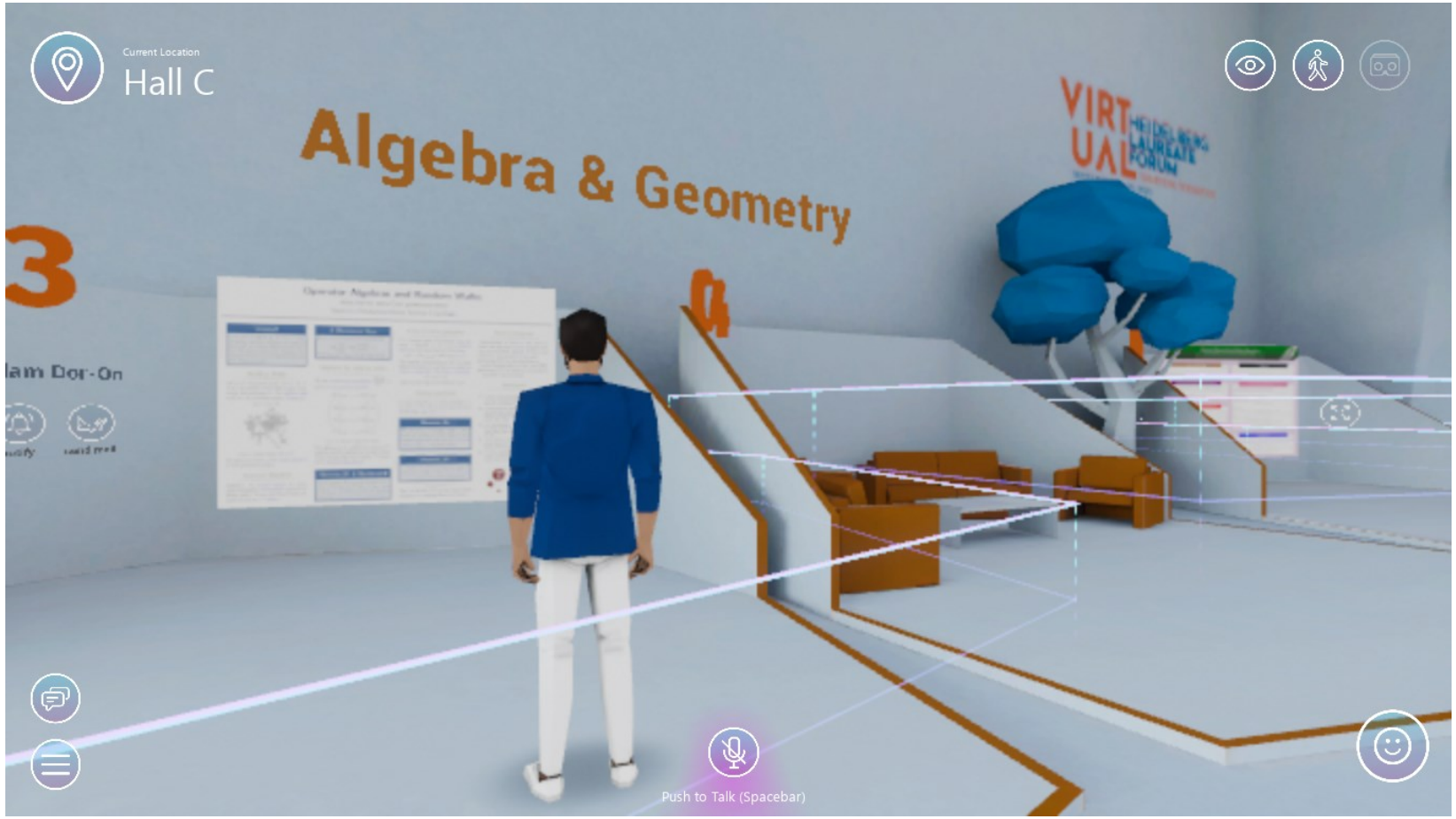
[Virtual HLF 2020 - Heidelberg Laureate Forum]

3D Virtual Event: Poster Sessions



Virtual HLF 2020 - Heidelberg Laureate Forum

3D Virtual Event: Poster Sessions



[Virtual HLF 2020 - Heidelberg Laureate Forum]

3D Virtual Event: Poster Sessions

Virtual HLF 2020 - Heidelberg Laureate Forum

3D Virtual Event: Virtual Gathering

Virtual HLF 2020 - Heidelberg Laureate Forum

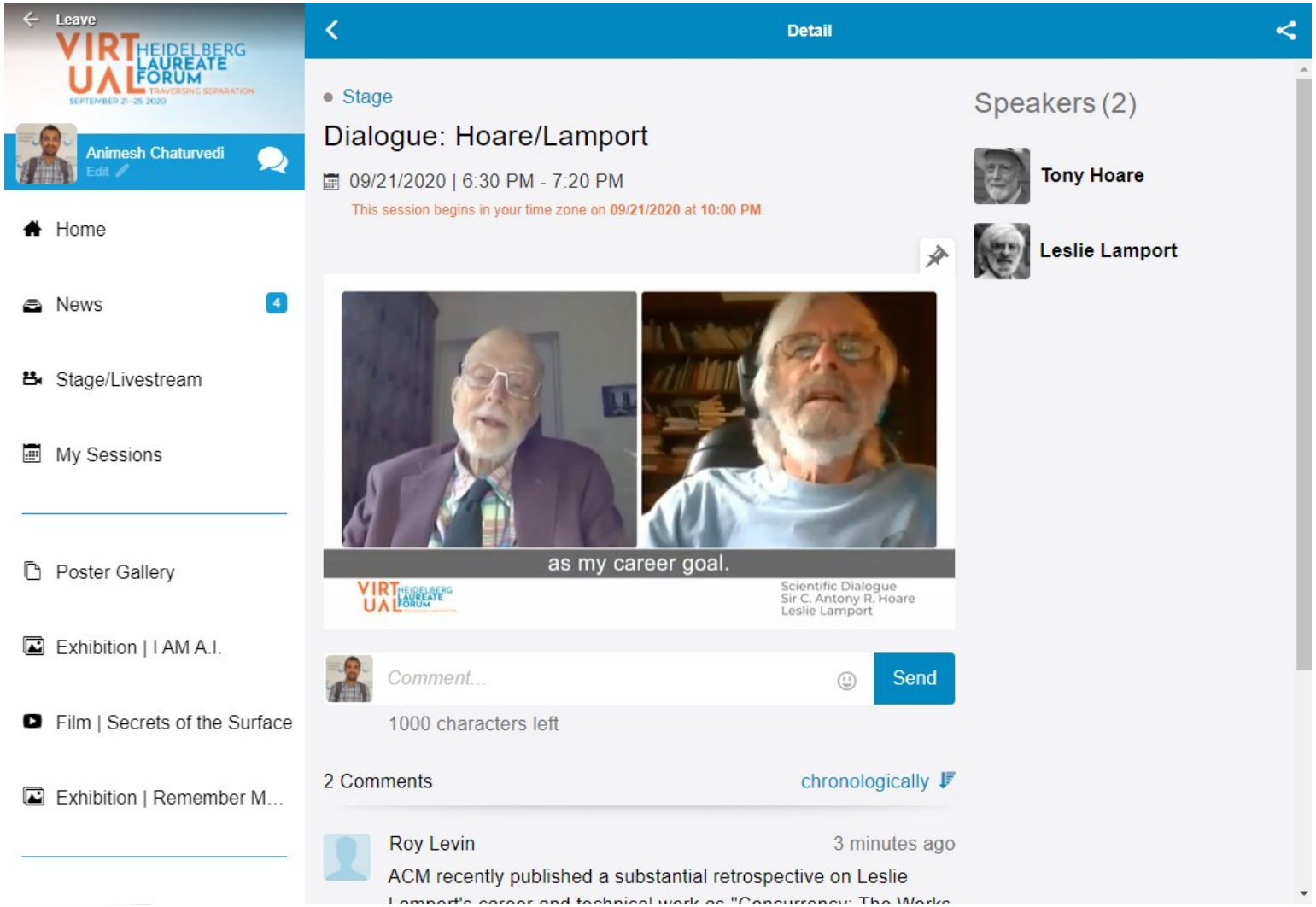
3D Virtual Event: Recreations

Virtual HLF 2020 - Heidelberg Laureate Forum

3D Virtual Event: Virtual Dancing

Virtual HLF 2020 - Heidelberg Laureate Forum

Live Streaming



[Virtual HLF 2020 - Heidelberg Laureate Forum]

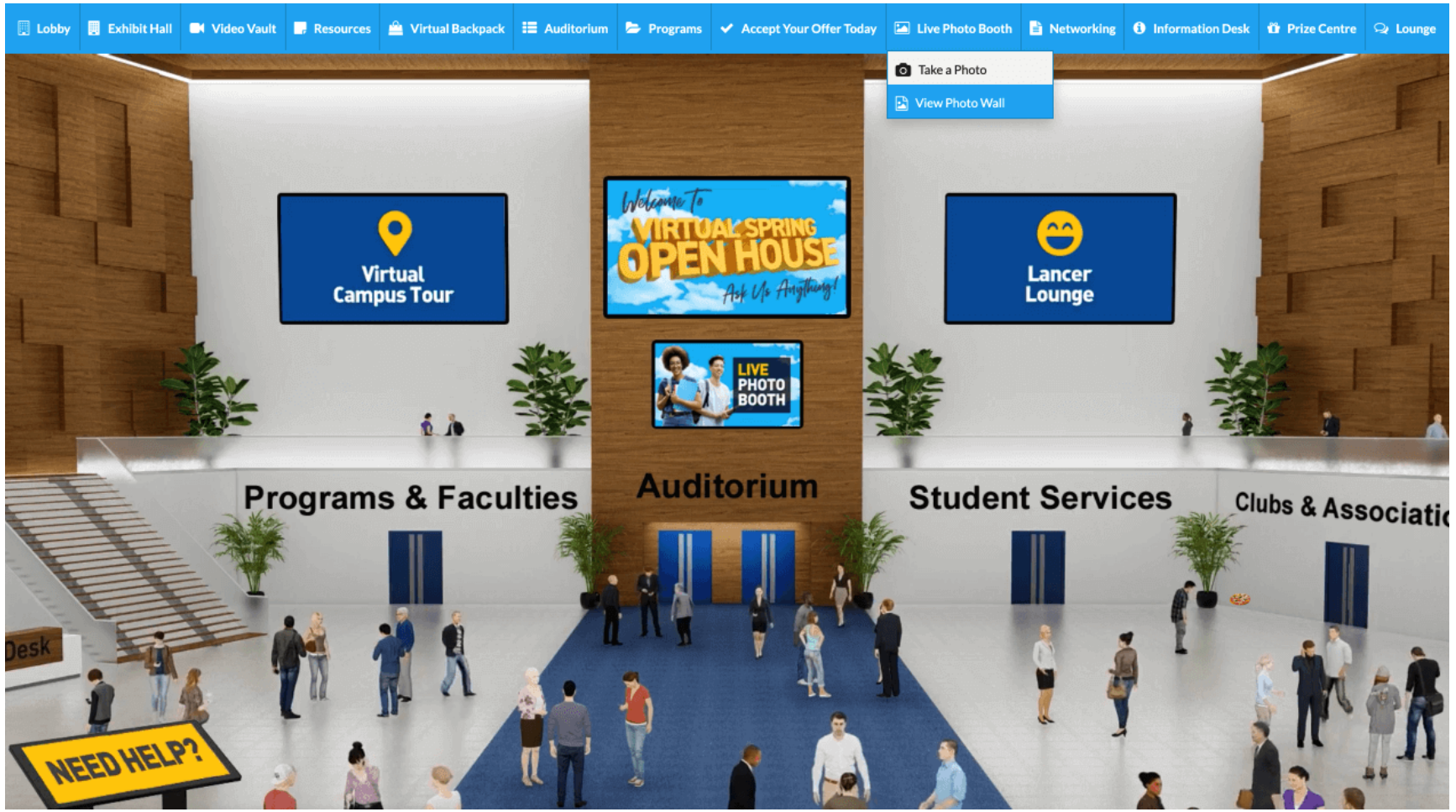
Hybrid Event

- Combines a “physical" in-person event with a "virtual" online component
- Tradeshow, Conference, Seminar, Workshop, Convocation

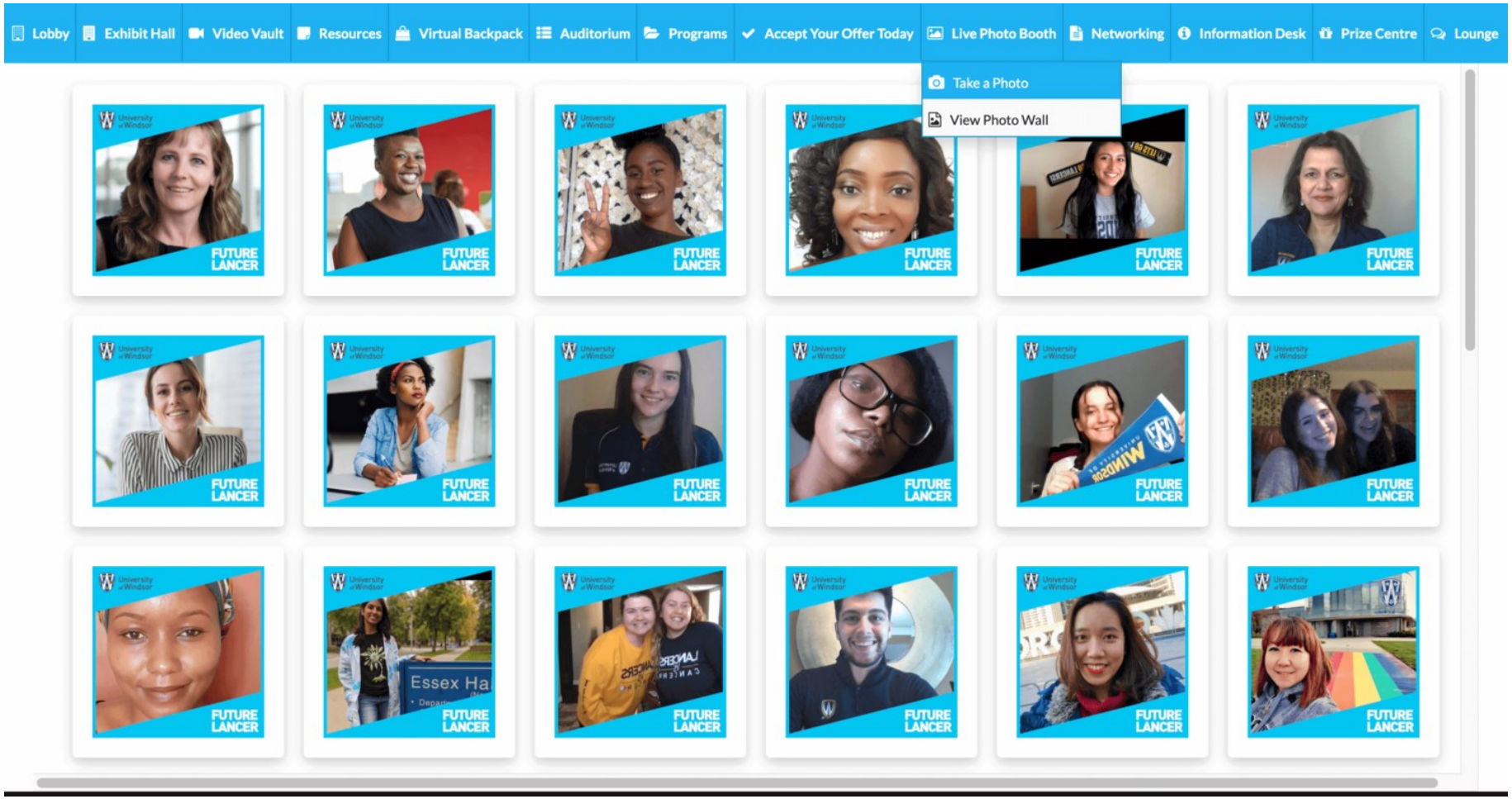
https://en.wikipedia.org/wiki/Hybrid_event

Hybrid Event: Landing Page

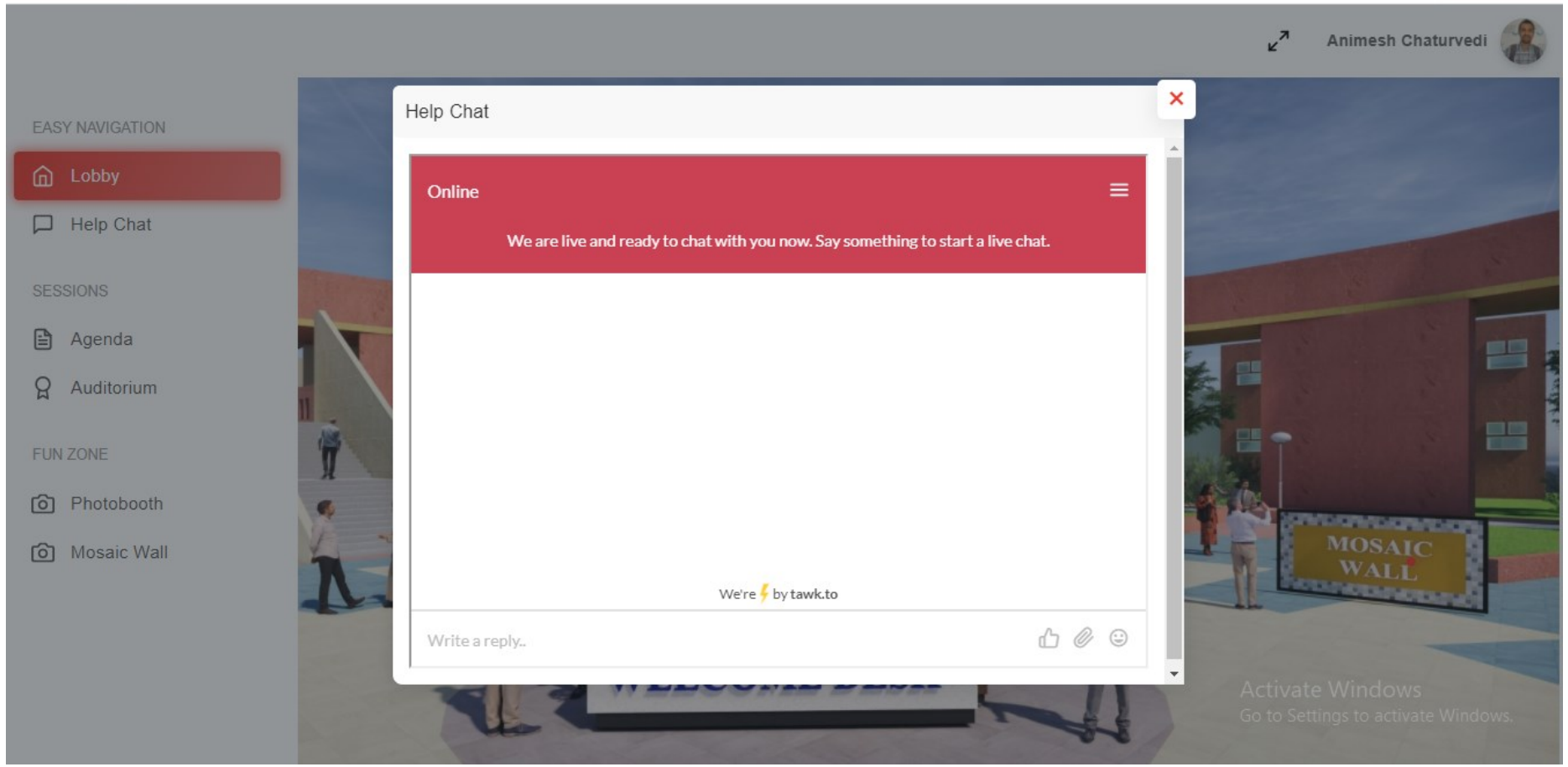
Hybrid Event: Landing Page



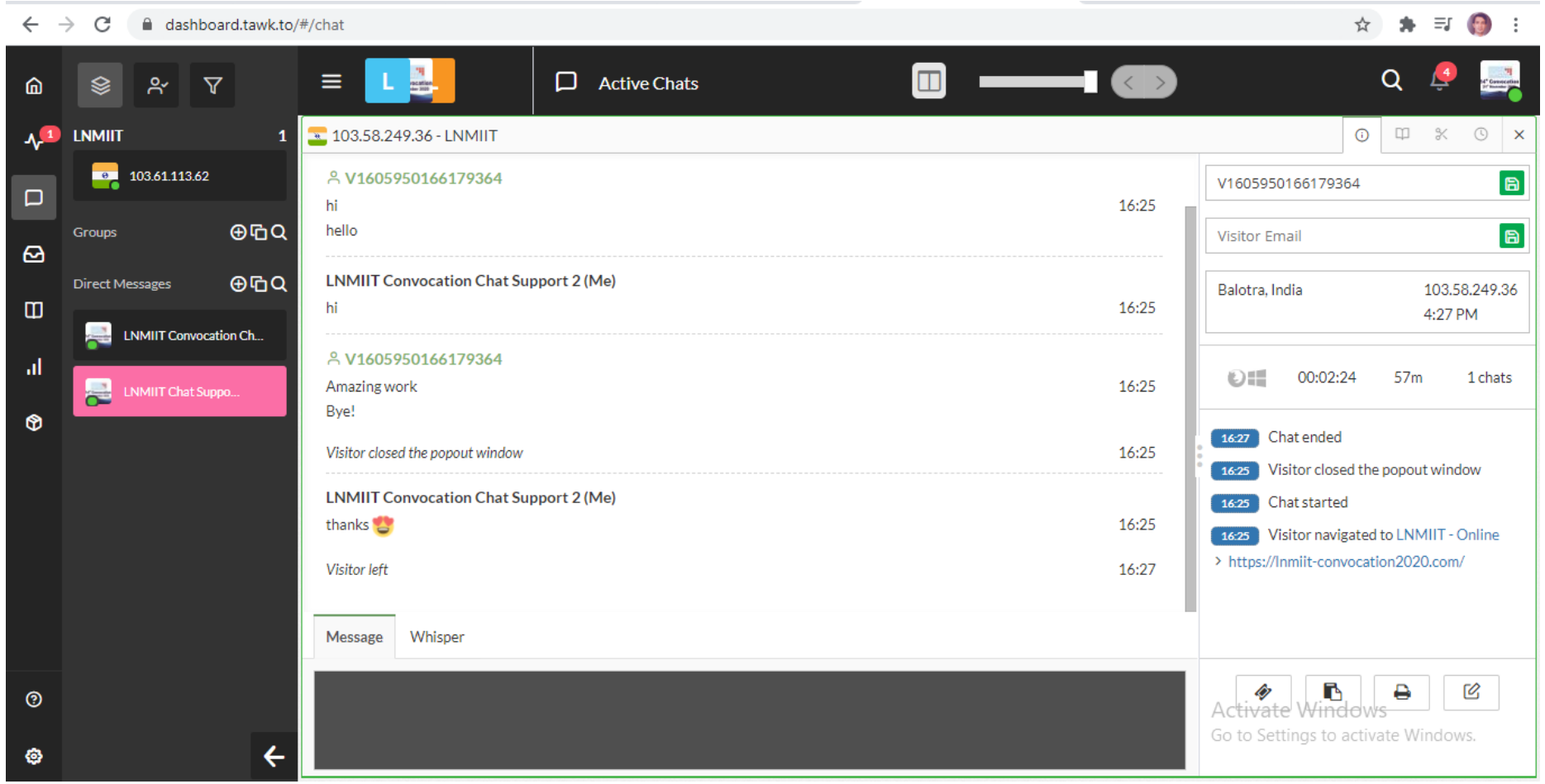
Hybrid Event: PhotoBooth



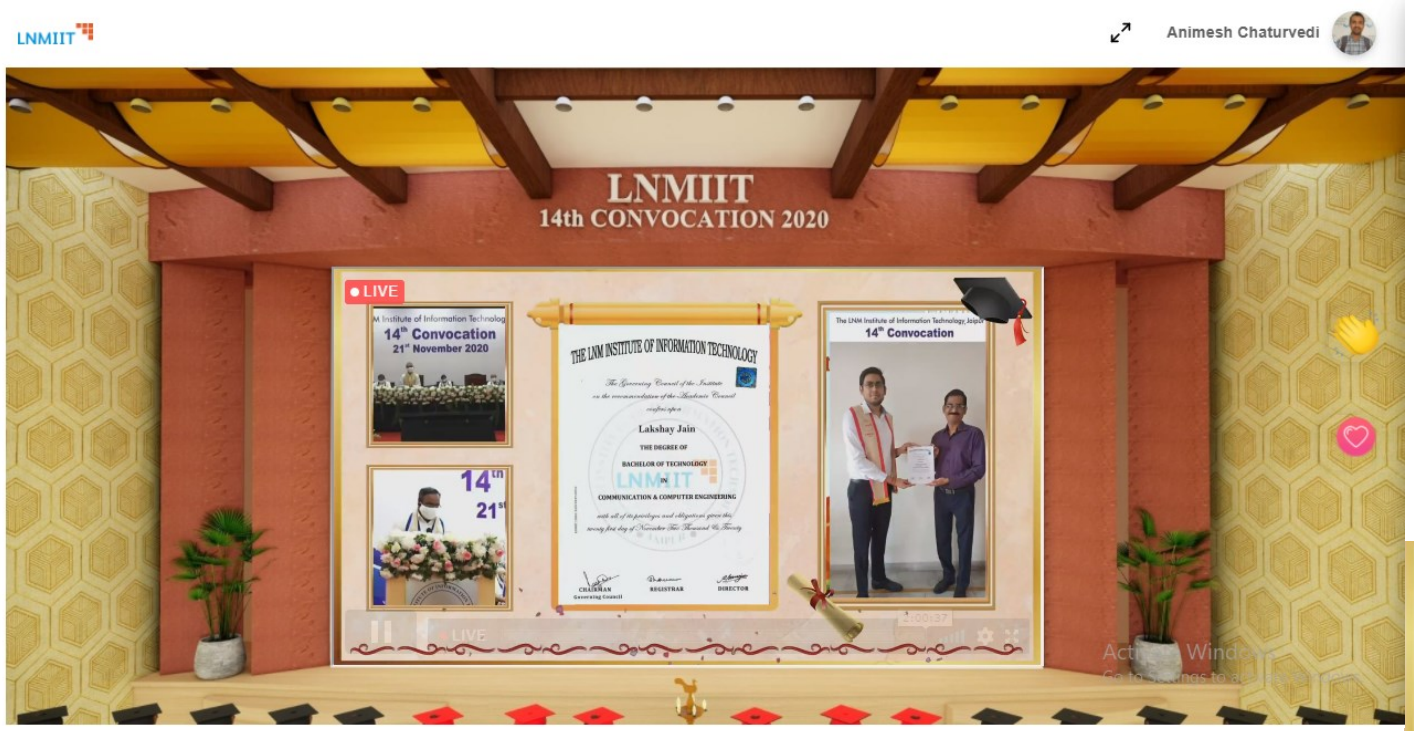
Hybrid Event: Help Chat FrontEnd



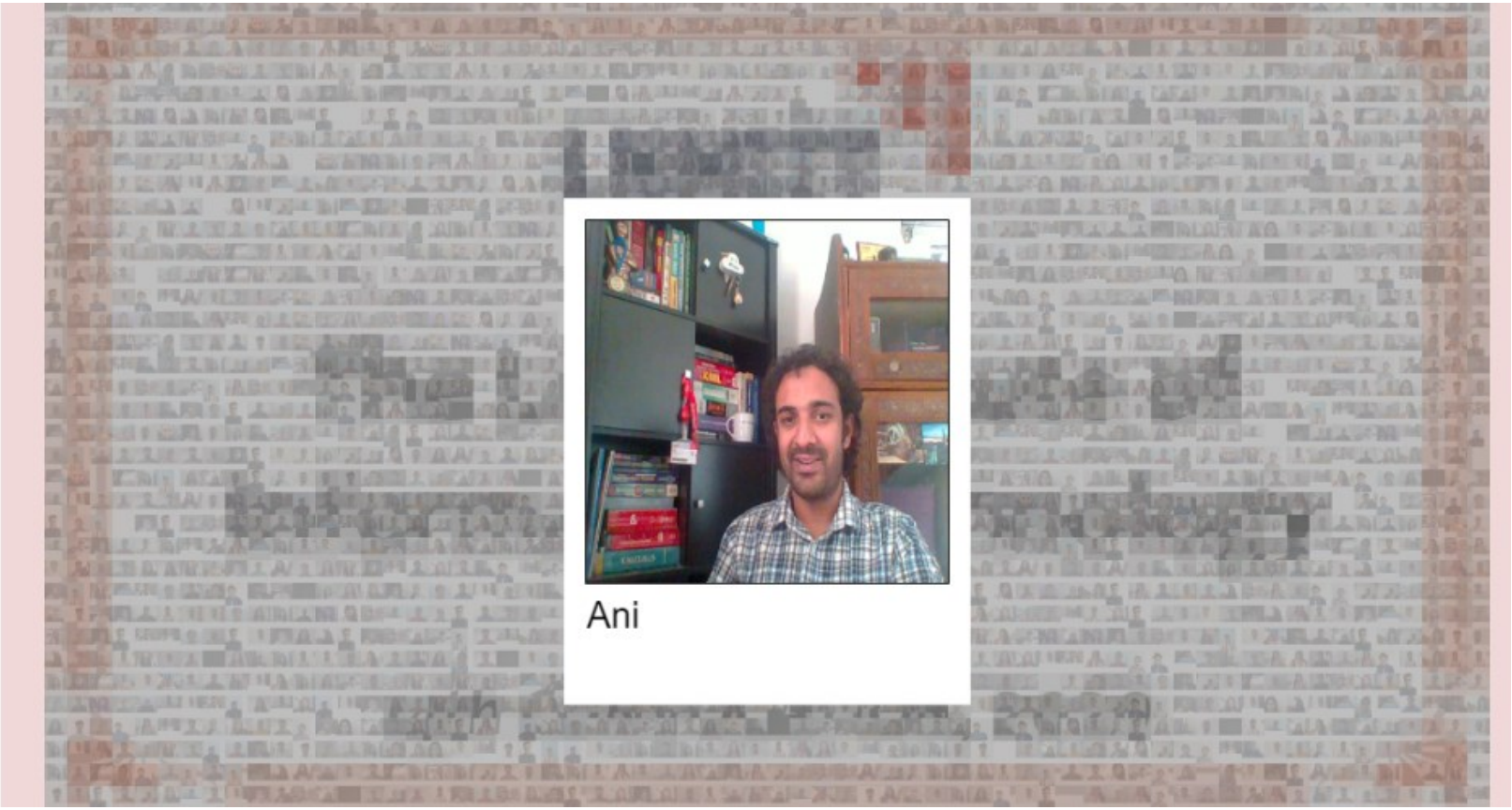
Hybrid Event: Help Chat BackEnd



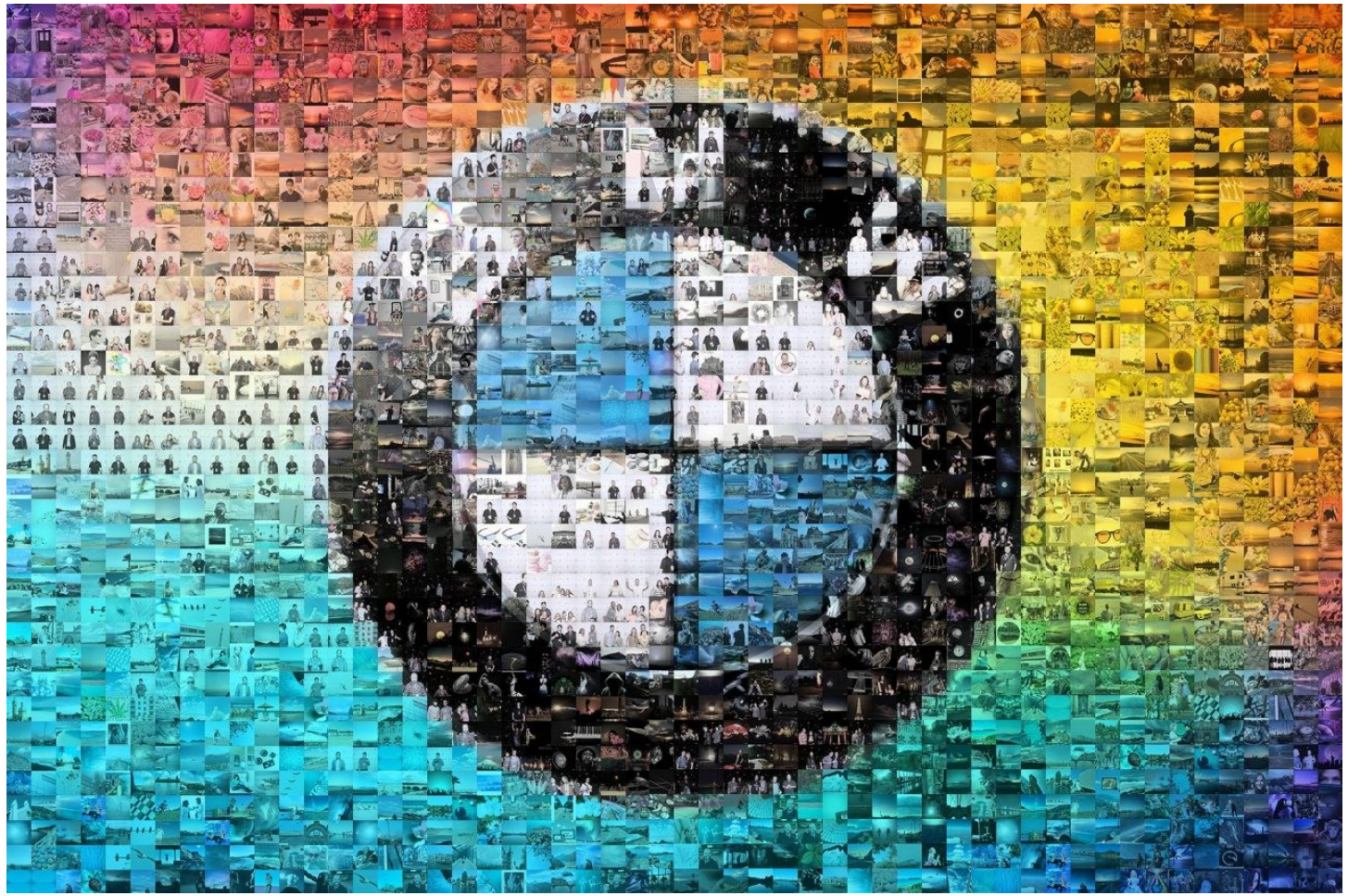
Hybrid Event: Virtual and Physical Degree



Hybrid Event: Mosaic Wall



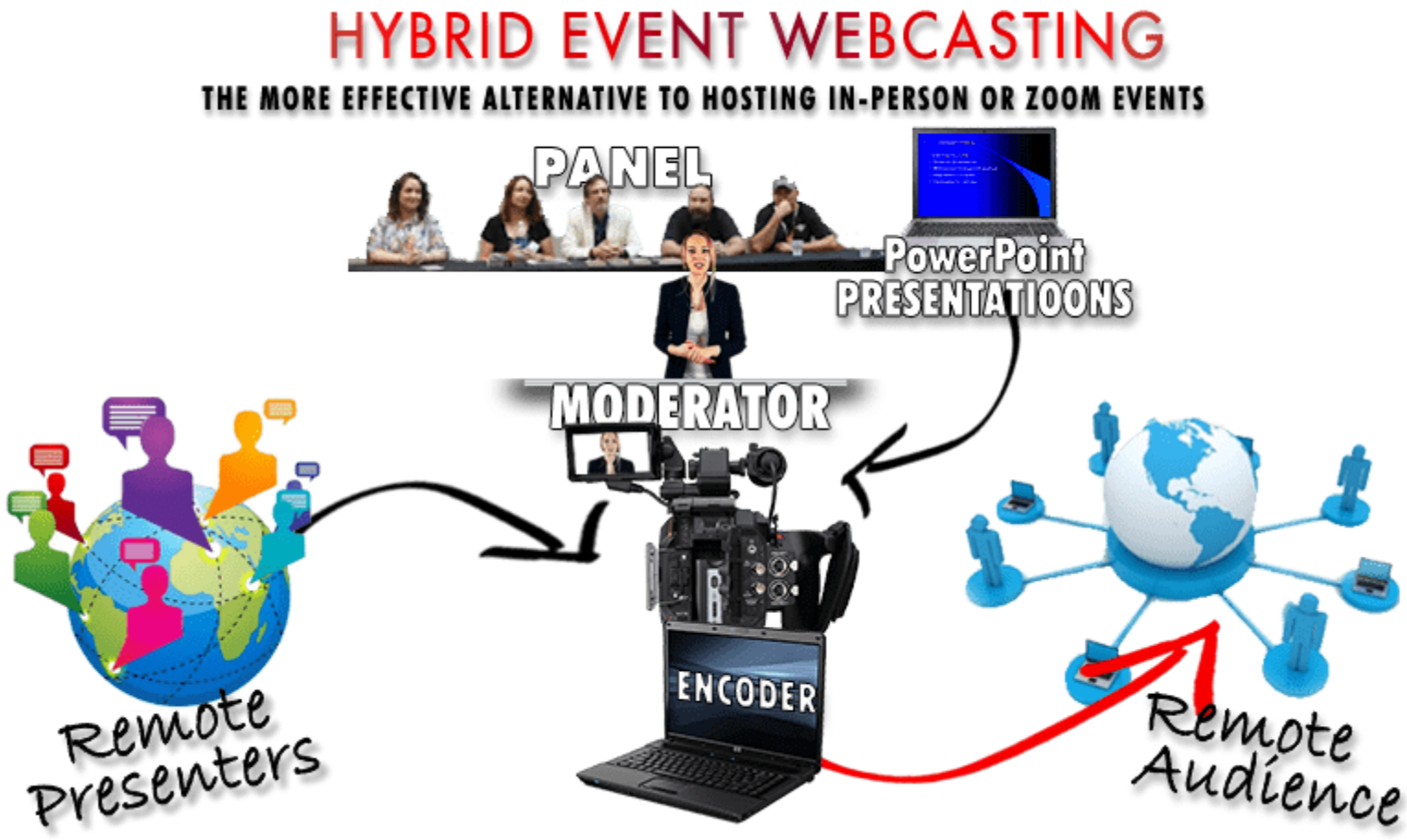
Hybrid Event: Mosaic Wall



https://www.youtube.com/watch?v=BSwJCRKvmPc

Hybrid Event: Blowing Flower Effect

Hybrid Event: Welcome WebCast



Hybrid Event: Virtual Address

Hybrid Event: Remote Connectivity

References

- https://spark.apache.org/
- Zaharia, Matei, et al. "Apache spark: a unified engine for big data processing." Communications of the ACM 59.11 (2016): 56-65.
- Zaharia, Matei, et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12). 2012.
- https://spark.apache.org/docs/latest/cluster-overview.html
- https://spark.apache.org/
- https://spark.apache.org/sql/
- https://spark.apache.org/streaming/
- https://spark.apache.org/graphx/
- Han Jiawei, Jian Pei, and Yiwen Yin. "Mining frequent patterns without candidate generation." *ACM SIGMOD Record* 29.2 (2000): 1-12.
- Li, Haoyuan, et al. "PFP: Parallel FP-Growth for query recommendation." *Proceedings of the 2008 ACM Conference on Recommender systems*. 2008.

References

- Weiser, Mark. "The computer for the 21st century." *ACM SIGMOBILE mobile computing and communications review* 3.3 (1999): 3-11.
- https://en.wikipedia.org/wiki/Ubiquitous_computing
- https://en.wikipedia.org/wiki/Edge_computing
- https://en.wikipedia.org/wiki/Cloudlet
- IEEE Standard Association. "IEEE 1934-2018-IEEE Standard for adoption of OpenFog reference architecture for fog computing." (2018).
- https://en.wikipedia.org/wiki/Fog_computing
- https://en.wikipedia.org/wiki/Virtual_event
- Virtual HLF 2020 - Heidelberg Laureate Forum
- https://en.wikipedia.org/wiki/Hybrid_event

ขอบคุณ
Thai

Grazie
Italian

תודה רבה
Hebrew

धन्यवादः
Sanskrit

ಧನ್ಯವಾದಗಳು
Kannada

Ευχαριστώ
Greek

Thank You
English

Gracias
Spanish

Спасибо
Russian

Obrigado
Portuguese

شكراً
Arabic

https://sites.google.com/site/animeshchaturvedi07

Merci
French

多謝
Traditional
Chinese

धन्यवाद
Hindi

Danke
German

多谢
Simplified
Chinese

நன்றி
Tamil
Tamil

ありがとうございました
Japanese

감사합니다
Korean