Advanced Data Processing Techniques for Distributed Applications and Systems

Hong-Linh Truong
Faculty of Informatics, TU Wien

hong-linh.truong@tuwien.ac.at
www.Infosys.tuwien.ac.at/staff/truong
@linhsolar
What this lecture is about?

- Large-scale data analytics
- Advanced messaging
  - Apache Kafka
- Advanced data analytics with streaming data processing
  - Main common features
  - Stream processing examples with Apache Apex
- Advanced data analytics with workflows
  - Data pipeline with Beam
  - Complex workflows with Airflow
Analytics-as-a-service

- **Goals**
  - Developers, Service Providers & Infrastructure Providers:
    - Understand and manage services systems
  - Service Providers:
    - Understand customers and optimize business

- **Examples**
  - Analyze monitoring information, logs, user activities, etc.
  - Predict usage trends for optimizing business

- **Techniques → Big data analytics**
  - Handle and process big data at rest and in motion
Key issues in large-scale data analytics

- Collect/produce messages from distributed application components and large-scale monitoring systems
  - Cross systems and cross layers
- Need scalable and reliable large-scale messaging broker systems
- Require workflow and stream data processing capabilities
- Integrate with various different types of services and data sources
Example from Lecture 4

- Multiple topics
- Amount of data per topic varies
- Should not have duplicate data in database

- Should I use docker? VMs?
- Where elasticity can be applied?
- Topic/data distribution to ingest clients?
Implementation atop Google cloud

Source: https://cloud.google.com/solutions/architecture/streamprocessing
Security-related information and metrics from distributed customers

Example: monitoring and security

Example: Bigdata analytics in SK Telco

Source: Yousun Jeong  https://www.slideshare.net/jerryjung7/stsg17-speaker-yousunjeong
Cloud services and big data analytics

Data sources (sensors, files, database, queues, log services)

Messaging systems (e.g., Kafka, AMQP, MQTT)

Storage and Database (S3, Google BigQuery, InfluxDB, HDFS, Cassandra, MongoDB, Elastic Search etc.)

Batch data processing systems (e.g., Hadoop, Airflow, Spark)

Stream processing systems (e.g., Apex, Kafka, Flink, WSO2, Google Dataflow)

Operation/Management/Business Services

Warehouse Analytics

Elastic Cloud Infrastructures (VMs, dockers, OpenStack elastic resource management tools, storage)
Recall: Message-oriented Middleware (MOM)

- Well-supported in large-scale systems for
  - Persistent and asynchronous messages
  - Scalable message handling
- Message communication and transformation
  - publish/subscribe, routing, extraction, enrichment
- Several implementations

- Amazon SQS
- MQTT
- Apache Kafka
- JMS
- CloudMQTT
- RabbitMQ
Recall: Workflow of Web services

- You learn it from the Advanced Internet Computing course

- Typically for composing Web services from different enterprises/departments for different tasks → many things have been changed in the cloud environment

- For big data analytics and Analytics-as-a-Service
  - Tasks are not just from Web services
http://kafka.apache.org/, originally from LinkedIn

APACHE KAFKA
Some use cases

- Producers generate a lot of realtime events
- Producers and consumers have different processing speeds
  - E.g. activity logging
- Rich and diverse types of events
  - E.g. cloud-based logging
- Dealing with cases when consumers might be on and off (fault tolerance support)

Which techniques can be used to control this?
More than message broker

- Messaging features
  - For transferring messages
    - Other frameworks in the ecosystem: RabbitMQ, Mostquito

- Streaming processing
  - Streaming applications handle data from streams
  - Read and write data back to Kafka messaging brokers
  - Other frameworks in the ecosystem: Apache Flink and Apache Apex

- High-level SQL-style: KSQL
  - Other possibilities: SQL-liked + Java in Apache Flink
Kafka Messaging
Kafka Design

- Use cluster of brokers to deliver messages
- A topic consists of different partitions
- Durable messages, ordered delivery via partitions
- Online/offline consumers
- Using filesystem **heavily** for message storage and caching
Messages, Topics and Partitions

- Ordered, immutable sequence of messages
- Messages are kept in a period of time (regardless of consumers or not)
- Support total order for messages within a partition
- Partitions are distributed among server

Source: http://kafka.apache.org/documentation.html
Consumers

- Consumer pulls the data
- The consumer keeps a single pointer indicating the position in a partition to keep track the offset of the next message being consumed

Why?
- allow customers to design their speed
- support/optimize batching data
- easy to implement total order over message
- easy to implement reliable message/fault tolerance
Example of a Producer

```java
public SimpleProducer( String url, String inputFile, String topic ) {
    Properties props = new Properties();
    props.put("bootstrap.servers", url);
    props.put("client.id", "rseea.io.training.demo");
    props.put("key.serializer", "org.apache.kafka.common.serialization.IntegerSerializer");
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
    producer = new KafkaProducer<Integer, String>(props);
    this.topic = topic;
    this.inputFile = inputFile;
}

public void run() {
    int messageNo = 1;
    // read data from file
    try {
        BufferedReader in = new BufferedReader(new FileReader(inputFile));
        Iterable<CSVRecord> records = CSVFormat.RFC4180.withFirstRecordAsHeader().parse(in);
        for (CSVRecord record : records) {
            JSONObject event = new JSONObject();
            event.addProperty("USERPHONE", 6645);
            event.addProperty("TIME", Long.parseLong(record.get("TIME")));
            event.addProperty("lat", Float.parseFloat(record.get("LATITUDE")));
            event.addProperty("lon", Float.parseFloat(record.get("LONGITUDE")));
            event.addProperty("GSM_BIT_ERROR_RATE", Float.parseFloat(record.get("GSM_BIT_ERROR_RATE")));
            event.addProperty("GSM_SIGNAL_STRENGTH", Float.parseFloat(record.get("GSM_SIGNAL_STRENGTH")));
            // a simple way to handle missing data is to skip the record
            if (!record.get("LOC_ACCURACY").equals("")) {
                event.addProperty("LOC_ACCURACY", Float.parseFloat(record.get("LOC_ACCURACY")));
            } else {
                continue;
            }
            if (!record.get("LOC_SPEED").equals("")) {
                event.addProperty("LOC_SPEED", Float.parseFloat(record.get("LOC_SPEED")));
            } else {
                continue;
            }
            String eventString = "{"event": " + event + "}";
            try {
                producer.send(new ProducerRecord<Integer, String>(topic, messageNo, eventString)).get();
            } catch (ExecutionException e) {
                // TODO: auto-generated catch block
                e.printStackTrace();
            }
        }
    }
```

DST 2018
public class SimpleConsumer {
    private final KafkaConsumer<Integer, String> consumer;
    private final String topic;
    private final int pollNr;
    public SimpleConsumer(String url, String topic, int pollNr) {
        Properties props = new Properties();
        // just use standard example configuration
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, url);
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "RDSEA Simple Consumer");
        props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
        props.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, "1000");
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, "30000");
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.IntegerDeserializer");
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");

        consumer = new KafkaConsumer<Integer, String>(props);
        this.topic = topic;
        this.pollNr = pollNr;
    }

    public void readData() {
        consumer.subscribe(Collections.singletonList(this.topic));
        ConsumerRecords<Integer, String> records = consumer.poll(pollNr);
        for (ConsumerRecord<Integer, String> record : records) {
            System.out.println("Received message: ", record.key(), ", ", record.value() + "); at offset ", record.offset());
        }
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        if (args.length < 3) {
            System.out.println("Usage: SimpleProducer kafka_broker topic nr");
            System.exit(0);
        }
        int pollNr = Integer.valueOf(args[2]);
        SimpleConsumer consumer = new SimpleConsumer(args[0], args[1], pollNr);
        consumer.readData();
    }
}
Message delivery

- Still remember message delivery guarantees?
  - At most once
  - At least once
  - Exactly once
What does it mean exactly one?

- Producer: Idempotent delivery → no duplicate entry in the log
- Transaction-like semantics: either message to ALL partition topics or not at all
- Consumer behavior management
Scalability and Fault Tolerance

- Partitions are distributed and replicated among broker servers
- Consumers are organized into groups
- Each message is delivered to a consumer instance in a group
- One partition is assigned to one consumer

http://kafka.apache.org/documentation.html#majordesignelements
Why partitions?
- Support scalability
  - enable arbitrary data types and sizes for a topic
  - enable parallelism in producing and consuming data

But partitions are replicated, why?
- For fault tolerance
Partition Replication

The leader handles all read and write requests

Source: http://de.slideshare.net/junrao/kafka-replication-apachecon2013
Consumer Group

- Consumer Group: a set of consumers
  - is used to support scalability and fault tolerance
  - allows multiple consumers to read a topic
- In one group: each partition is consumed by only consumer instance
  - Combine „queueing“ and „publish/subscribe“ model
- Enable different applications receive data from the same topic.
  - different consumers in different groups can retrieve the same data
Group rebalancing

Key Questions/Thoughts

- Why do we need partitions per topic?
  ➔ arbitrary data handling, ordering guarantees, load balancing
- How to deal with high volume of realtime events for online and offline consumers?
  ➔ partition, cluster, message storage, batch retrieval, etc.
- Queuing or publish-subscribe model?
  ➔ check how Kafka delivers messages to consumer instances/groups
Kafka vs RabbitMQ

STREAMING DATA PROCESSING
Batch, Stream and Interactive Analytics

Batch – Ad-hoc queries on large data sets. I/O Bound
Interactive – Querying historical data
Real Time Streaming

Source: https://dzone.com/refcardz/apache-spark
Recall: Centralized versus distributed processing topology

Two views: **streams of events or cloud of events**

Complex Event Processing  
(centralized processing)

- Event cloud
- Processing
- node
- node
- node

Usually only queries/patterns are written

Streaming Data Processing  
(distributed processing)

- Processing
- node
- Processing
- node
- Processing
- node

Code processing events and topologies need to be written
• Data source operator: represents a source of streams
• Compute operators: represents processing functions
• Native versus micro-batching
Key concepts

- Structure of the data processing
  - Topology: Directed Acycle Graph (DAG) of operators
  - Data input/output operators and compute operators
  - Accepted various data sources through different connectors

- Scheduling and execution environments
  - Distributed tasks on multiple machines
  - Each machine can run multiple tasks

- Stream: connects an output port from an operator to an input port to another operator

- Stream data is sliced into windows of data for compute operators
Implementations

- Many implementation, e.g.
  - Apache Storm
    - https://storm.apache.org/
  - Apache Spark
    - https://spark.apache.org/
  - Apache Apex
    - https://apex.apache.org/
  - Apache Kafka and Apache Flink

Check:
http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processingFrameworks-part-1

Key common concepts

- Abstraction of streams
- Connector library
  - Very important for application domains
- Runtime elasticity
  - Add/remove (new) operators (and underlying computing node)
- Fault tolerance
Recall:

Data stream: a sequence/flow of data units
Data units are defined by applications: a data unit can be data described by a primitive data type or by a complex data type, a serializable object, etc.

In Apache Apex: a stream of atomic data elements (tuples)

In Apache Kafka: data element is <Key,Value> tuple
Example of an Apex application in Java

```java
@ApplicationAnnotation(name="MySecondApplication")
public class BTSApplication implements StreamingApplication {
    String topic = "apextest";
    QoS qos;

    public BTSApplication() {
        this.qos = QoS.AT_MOST_ONCE;
    }

    @Override
    public void populateDAG(DAG dag, Configuration conf) {
        System.out.println("Start the application by connecting to MQTT.");
        MqttClientConfig btsmqttConfig = new MqttClientConfig();
        btsmqttConfig.setHost("localhost");
        btsmqttConfig.setPort(1883);
        btsmqttConfig.setUserName("guest");
        btsmqttConfig.setPassword("guest");
        btsmqttConfig.setCleanSession(true);

        // creating input operator
        VietcontrolMQTTInput btsInput = dag.addOperator("input", VietcontrolMQTTInput.class);
        btsInput.setMqttClientConfig(btsmqttConfig);
        System.out.println("Subscribe topics");
        btsInput.addSubscribeTopic(topic, qos);

        // just a simple example to output the data to the console
        ConsoleOutputOperator cons = dag.addOperator("console", new ConsoleOutputOperator());
        cons.setSilent(false);
        System.out.println("Just create one single stream");
        dag.addStream("test", btsInput.out, cons.input).setLocality(Locality.CONTAINER_LOCAL);
    }

    public class VietcontrolMQTTInput extends AbstractMqttInputOperator{
        public final transient DefaultOutputPort<String> out;

        public VietcontrolMQTTInput() {
            this.out = new DefaultOutputPort<>();
            //out.emit("Test message");
        }
    }
}
```
Processor/Operators

- Streaming applications are built with a set of processors/operators: for data and computation

- Some common data operators (related to other lectures)
  - MQTT
  - AMQP
  - Kafka

Source: https://apex.apache.org/docs/malhar/
Why are the richness and diversity of connectors important?
Time and stream processing

Can you explain the time notion and the roles?
Fault tolerance

- Recovery
  - At least once
  - At most once
  - Exactly once
  - E.g. Kafka Streams: Exactly once and at least once

- Note the difference between messaging and processing w.r.t fault tolerance
Some (interesting) features of Apache Apex
DAG of Operators

- **Ports**: for input and output data
- **Data in a stream**: streaming windows

Source: https://apex.apache.org/docs/apex-3.6/operator_development/
Processing data in operators

Different types of Windows: GlobalWindows, TimeWindows, SlidingTimeWindows, etc.

Source: https://apex.apache.org/docs/apex/operator_development/
Execution Management

- Using YARN for execution tasks
- Using HDFS for persistent state
Understand YARN/Hadoop to understand Apex operator execution management

Source: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
Scalability

- Locality configuration for deployment of streams and operators
- Affinity and anti-affinity rules
- Possible localities:
  - THREAD_LOCAL (intra-thread)
  - CONTAINER_LOCAL (intra-process)
  - NODE_LOCAL (inter-process but within a Hadoop node)
  - RACK_LOCAL (inter-node)
Operators Fault tolerance

- Checkpoint of operators: save state of operators (e.g. into HDFS)
  - @Stateless no checkpoint
  - Check point interval: CHECKPOINT_WINDOW_COUNT

- Recovery
  - At least once
  - At most once
  - Exactly once
Example of Partitioning and unification in Apex

- Dynamic Partition
  - Partition operators
  - Dynamic: specifying when a partition should be done
  - Unifiers for combining results (reduce)

- StreamCodec
  - For deciding which tuples go to which partitions
  - Using hashcode and masking mechanism

Source:
https://apex.apache.org/docs/apex/application_development/#partitioning
Fault tolerance – Recovery in Apex

- At least once
  - Downstream operators are restarted
  - Upstream operators are replayed
- At most once
  - Assume that data can be lost: restart the operator and subscribe to new data from upstream
- Exactly once
Exercise

How to make sure no duplication results when we recover End-to-End Exactly Once?

How to use hash and masking mechanism to distributed tuples?

How to deal with data between operators not in a CONTAINER_LOCAL or in THREAD_LOCAL
Use cases

- Access and coordinate many different compute services, data sources, deployment services, etc, within an enterprise, for a particular goal
- Implementing complex “business logics“ of your services
- Analytics-as a service: metrics, user activities analytics, testing, e.g.
  - Analytics of log files (generated by Aspects in Lecture 3)
  - Dynamic analytics of business activities
Workflow and Pipeline/data workflow

- Workflows: a set of coordinated activities
  - Generic workflows of different categories of tasks
  - Data workflows → data pipeline
    
    „a pipeline is a set of data processing elements connected in series, where the output of one element is the input of the next one”

    Source: [https://en.wikipedia.org/wiki/Pipeline_%28computing%29](https://en.wikipedia.org/wiki/Pipeline_%28computing%29)

- We use a pipeline/data workflows to carry out a data processing job
Pipeline p = Pipeline.create(options);

// Concepts #2 and #3: Our pipeline applies the composite CountWords transform, and passes the
// static FormatAsTextFn() to the ParDo transform.

p.apply("ReadLines", TextIO.read().from(options.getInputFile()))
  .apply(new CountWords())
  .apply(MapElements.via(new FormatAsTextFn()))
  .apply("WriteCounts", TextIO.write().to(options.getOutputStream()));

p.run().waitUntilFinish();

Example with Node-RED

Node-RED
Flow-based programming for the Internet of Things

Figure source: Hong-Linh Truong, Enabling Edge Analytics of IoT Data: the Case of LoRaWAN, The 2018 Global IoT Summit (GIoTS) 4-7 June 2018 in Bilbao, Spain
Data analytics workflow execution models
Workflow and Pipeline/data workflow

- But analytics have many more than just data processing activities
  - Storage: where is the data from? Where is the sink of data?
  - Communication of results:
    - is software or human the receiver of the analytics results?:
    - Software: messaging, serverless function, REST API, Webhook?
    - People: Email, SMS, ...
  - Visualization of results: which tools?
Your are in a situation:

- Many underlying distributed processing frameworks
  - Apex, Spark, Flink, Google
- Work with different underlying engines
- Write only high-level pipelines
- Stick to your favour programming languages
Apache Beam

- **Goal:** separate from pipelines from backend engines

  - Read data analytics
  - Post-processing result
  - Store analysis result

- Technologies:
  - Apache Apex™
  - Flink
  - Apache Spark™
  - Dataflow
Apache Beam

- https://beam.apache.org/
- Suitable for data analysis processes that can be divided into different independent tasks
  - ETL (Extract, Transform and Load)
  - Data Integration
- Execution principles:
  - Mapping tasks in the pipeline to concrete tasks that are supported by the selected back-end engine
  - Coordinating task execution like workflows.
Basic programming constructs

- **Pipeline:**
  - For creating a pipeline

- **PCollection**
  - Represent a distributed dataset

- **Transform**
  
  \[
  \text{[Output PCollection]} = \text{[Input PCollection]} \mid \text{[Transform]}
  \]

- Possible transforms: ParDo, GroupByKey, Combine, etc.
A simple example with Google Dataflow as back-end engine

```python
import apache_beam as beam
from apache_beam.options.pipeline_options import PipelineOptions

p = beam.Pipeline(options=PipelineOptions())

entries = p | 'ReadHadoopResult' >> beam.io.ReadFromText('gs://.../ElectricityAlarm/electricity_alarm_frequency-2017-05-11-00-vn.csv')

class ExtractAlarmFrequency(beam.DoFn):
    def process(self, elements):
        ....
        return ....

frequency = entries | beam.ParDo(ExtractAlarmFrequency())
frequency | 'write' >> beam.io.WriteToText('gs://.../ElectricityAlarm')

result = p.run()
result.wait_until_finish()
```
Beam SQL

- https://beam.apache.org/documentation/dsls/sql/
- High level SQL-like statements
- Combine with Java APIs
- Common features
  - Aggregation functions
  - Windows
  - User-defined functions
But what if you need diverse types of tasks with various back-end services?

→ Workflow systems
Example of using workflows

Representing and programming workflows/data workflows

- Programming languages
  - General- and specific-purpose programming languages, such as Java, Python, Swift

- Descriptive languages
  - BPEL and several languages designed for specific workflow engines
Key requirements for us in the Cloud

- Rich connectors to various data sources
- Computation engines
- Different underlying infrastructures
- REST and message broker integration
Example with Apache Airflow
Airflow from Airbnb


**Features**
- Dynamic, extensible, scalable workflows
- Programmable language based workflows
  - Write workflows as programmable code
- Good and easy to study to understand concepts of workflows/data pipeline
Many connectors

<table>
<thead>
<tr>
<th>subpackage</th>
<th>install command</th>
<th>enables</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>pip install apache-airflow[all]</td>
<td>All Airflow features known to man</td>
</tr>
<tr>
<td>all_dbs</td>
<td>pip install apache-airflow[all_dbs]</td>
<td>All databases integrations</td>
</tr>
<tr>
<td>async</td>
<td>pip install apache-airflow[async]</td>
<td>Async worker classes for gunicorn</td>
</tr>
<tr>
<td>devel</td>
<td>pip install apache-airflow[devel]</td>
<td>Minimum dev tools requirements</td>
</tr>
<tr>
<td>devel_hadoop</td>
<td>pip install apache-airflow[devel_hadoop]</td>
<td>Airflow + dependencies on the Hadoop</td>
</tr>
<tr>
<td>celery</td>
<td>pip install apache-airflow[celery]</td>
<td>CeleryExecutor</td>
</tr>
<tr>
<td>crypto</td>
<td>pip install apache-airflow[crypto]</td>
<td>Encrypt connection passwords in metastore</td>
</tr>
<tr>
<td>druid</td>
<td>pip install apache-airflow[druid]</td>
<td>Druid.io related operators &amp; hooks</td>
</tr>
<tr>
<td>gcp_api</td>
<td>pip install apache-airflow[gcp_api]</td>
<td>Google Cloud Platform hooks and operators</td>
</tr>
<tr>
<td>jdbc</td>
<td>pip install apache-airflow[jdbc]</td>
<td>JDBC hooks and operators</td>
</tr>
<tr>
<td>hdfs</td>
<td>pip install apache-airflow[hdfs]</td>
<td>HDFS hooks and operators</td>
</tr>
<tr>
<td>hive</td>
<td>pip install apache-airflow[hive]</td>
<td>All Hive related operators</td>
</tr>
<tr>
<td>kerberos</td>
<td>pip install apache-airflow[kerberos]</td>
<td>kerberos integration for kerberized had</td>
</tr>
<tr>
<td>ldap</td>
<td>pip install apache-airflow[ldap]</td>
<td>Ldap authentication for users</td>
</tr>
<tr>
<td>mssql</td>
<td>pip install apache-airflow[mssql]</td>
<td>Microsoft SQL operators and hook, support</td>
</tr>
<tr>
<td>mysql</td>
<td>pip install apache-airflow[mysql]</td>
<td>MySQL operators and hook, support</td>
</tr>
<tr>
<td>password</td>
<td>pip install apache-airflow[password]</td>
<td>Password Authentication for users</td>
</tr>
<tr>
<td>postgres</td>
<td>pip install apache-airflow[postgres]</td>
<td>Postgres operators and hook, support</td>
</tr>
<tr>
<td>qds</td>
<td>pip install apache-airflow[qds]</td>
<td>Enable QDS (qubole data services) support</td>
</tr>
<tr>
<td>rabbitmq</td>
<td>pip install apache-airflow[rabbitmq]</td>
<td>Rabbitmq support as a Celery backend</td>
</tr>
<tr>
<td>s3</td>
<td>pip install apache-airflow[s3]</td>
<td>S3KeySensor, S3PrefixsSensor</td>
</tr>
<tr>
<td>samba</td>
<td>pip install apache-airflow[samba]</td>
<td>Hive2SambaOperator</td>
</tr>
<tr>
<td>slack</td>
<td>pip install apache-airflow[slack]</td>
<td>SlackAPIPostOperator</td>
</tr>
<tr>
<td>vertica</td>
<td>pip install apache-airflow[vertica]</td>
<td>Vertica hook support as an Airflow backend</td>
</tr>
<tr>
<td>cloudant</td>
<td>pip install apache-airflow[cloudant]</td>
<td>Cloudant hook</td>
</tr>
<tr>
<td>redis</td>
<td>pip install apache-airflow[redis]</td>
<td>Redis hooks and sensors</td>
</tr>
</tbody>
</table>

DST 2018
Airflow Workflow structure

- Workflow is a DAG (Direct Acyclic Graph)
  - A workflow consists of a set of activities represented in a DAG
  - Workflow and activities are programmed using Python – described in code
- Workflow activities are described by Airflow operator objects
  - Tasks are created when instantiating operator objects
Airflow from Airbnb

- Rich set of operators
  - So that we can program different kinds of tasks and integrate with different systems

- Different Types of operators for workflow activities
  - BashOperator, PythonOperator, EmailOperator, HTTPOperator, SqlOperator, Sensor,
  - DockerOperator, HiveOperator, S3FileTransferOperator, PrestoToMysqlOperator, SlackOperator
Example for processing signal file

Airflow GUI

worker

Computing servers

download_signal_file
analyticsinternetusage
sendresults

Localfile system (e.g. /opt/data/airflow)

Blob File Server

Notification Service (REST, NodeJS)

MQTT broker

ElasticSearch

UserApp

DST 2018
Example for processing signal file

```python
DAG_NAME = 'signal_upload_file'

default_args = {
    'owner': 'hong-lich-truong',
    'depends_on_past': False,
    'start_date': datetime.now(),
}

dag = DAG(DAG_NAME, schedule_interval=None, default_args=default_args)

stations=['station1', 'station2']

def checkSituation(**kwargs):
    f = 'f'
    t = 't'
    return t

downloadLogscript="curl file:///home/truong/myprojects/mgit/rdsea-mobifone-training/data/opensignal/sample-Oct182016.csv -o /opt/data/air"

t_downloadlogcloud= BashOperator(
    task_id="download_signal_file",
    bash_command=downloadLogscript,
    dag = dag
)

t_analytics= BashOperator(
    task_id="analyticsinternetusage",
    bash_command="/user/bin/python /home/truong/myprojects/mgit/rdsea-mobifone-training/examples/databases/elasticsearchuploader/src/uploaders/
    dag = dag
)

t_sendresult = SimpleHttpOperator(
    task_id='sendresults',
    method='POST',
    http_conn_id='station1',
    endpoint='api/update/credit',
    data=json.dumps({"userphone": "066412345","credit":10}),
    headers={"Content-Type": "application/json"},
    dag = dag
)

t_analytics.set_upstream(t_downloadlogcloud)
t_sendresult.set_upstream(t_analytics)
```
### Examples

#### DAGs

<table>
<thead>
<tr>
<th>#</th>
<th>DAG</th>
<th>Schedule</th>
<th>Owner</th>
<th>Recent Statuses</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>example_bash_operator</td>
<td>0 0 * * *</td>
<td>airflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>example_branch_dop_operator_v3</td>
<td>*/3 * * *</td>
<td>airflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>example_branch_operator</td>
<td>@daily</td>
<td>airflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>example_http_operator</td>
<td>1 day, 0:00:00</td>
<td>airflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>example_passing_params_via_test_command</td>
<td>*/3 * * *</td>
<td>airflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>example_python_operator</td>
<td>None</td>
<td>airflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>example_short_circuit_operator</td>
<td>1 day, 0:00:00</td>
<td>airflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>example_skip_dag</td>
<td>1 day, 0:00:00</td>
<td>airflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>example_subdag_operator</td>
<td>@once</td>
<td>airflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>example_trigger_controller_dag</td>
<td>@once</td>
<td>airflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>example_trigger_target_dag</td>
<td>None</td>
<td>airflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>example_trigger_target_dag</td>
<td>@daily</td>
<td>Ekhtiar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>example_twitter_dag</td>
<td>@once</td>
<td>airflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>example_xcom</td>
<td>@once</td>
<td>airflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>signal_upload_file</td>
<td>None</td>
<td>hong-linh-truong</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>tutorial</td>
<td>1 day, 0:00:00</td>
<td>airflow</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Showing 1 to 15 of 15 entries
Elasticity control for Workflows/Data Flows

- How to scale the workflows?
- Scheduling in a large resource pool (e.g., using clusters)
- Elasticity controls of virtualized resources (VMs/containers) for executing tasks
- Distributed Task Queue, e.g. Celery


Job description/request sent via queues
Results from jobs can be stored in some back-end
Other systems, e.g., AWS Data Pipeline

Hybrid service design

- Stream analytics triggers datapipes?
- Stream analytics triggers workflows?
- Stream analytics triggers serverless functions?
- And another way around?
Communicating results

- How to communicate results to the end user or other components?
- Software integration with protocols and interactions in previous lectures
- People: conversational commerce
  - More than just using SendGrid, Applozic, etc.

Here are examples of Duplex making phone calls (using different voices):

- Duplex scheduling a hair salon appointment:
- Duplex calling a restaurant:

Source: https://developer.amazon.com/alexa-voice-service

Source: https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html
Summary

- Analytics-as-a-service for large-scale distributed applications and big data analytics require different set of tools
- Kafka, Apache Apex and Airflow are just some of the key frameworks
  - There are a lot of tools
- Need to understand common concepts and distinguishable features
- Select them based on your use cases and application functionality and performance requirements
- Exercises:
  - a small application utilizing Kafka/MQTT and Apache Apex
  - Log analytics using AOP and Kafka and Airflow
Further materials

- http://kafka.apache.org
- https://cloud.google.com/dataflow/docs/
- http://storm.apache.org/
- https://dzone.com/articles/kafka-clients-at-most-once-at-least-once-exactly-o
Thanks for your attention

Hong-Linh Truong
Faculty of Informatics, TU Wien
hong-linh.truong@tuwien.ac.at
http://www.infosys.tuwien.ac.at/staff/truong
@linhsolar