Stream Data Processing

Data Sources
- Events
- Logs
- Sensor Data
- Social Databases
- CDC

Data Delivery
- kafka
- Amazon Kinesis
- SQS
- RabbitMQ
- Solace Systems
- nifi

Transform / Analytics
- Declarative API
- SQL
- DAG API
- Operator Library
- Beam
- SAMOA
- Oper1
- Oper2
- Oper3

Real-time visualization, …

Transform / Analytics
- APEX

Data Delivery
- APEX
- Hadoop
- Cloudera
- MapR
- Hortonworks
- Apache Mesos
- Oracle
- MySQL
- Cassandra
- Hbase
- MemSQL
- Apache Nifi
- APEX (roadmap)
# Industries & Use Cases

<table>
<thead>
<tr>
<th>Financial Services</th>
<th>Ad-Tech</th>
<th>Telecom</th>
<th>Manufacturing</th>
<th>Energy</th>
<th>IoT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fraud and risk monitoring</td>
<td>Real-time customer facing dashboards on key performance indicators</td>
<td>Call detail record (CDR) &amp; extended data record (XDR) analysis</td>
<td>Supply chain planning &amp; optimization</td>
<td>Smart meter analytics</td>
<td>Data ingestion and processing</td>
</tr>
<tr>
<td>Credit risk assessment</td>
<td>Click fraud detection</td>
<td>Understanding customer behavior AND context</td>
<td>Preventive maintenance</td>
<td>Reduce outages &amp; improve resource utilization</td>
<td>Predictive analytics</td>
</tr>
<tr>
<td>Improve turn around time of trade settlement processes</td>
<td>Billing optimization</td>
<td>Packaging and selling anonymous customer data</td>
<td>Product quality &amp; defect tracking</td>
<td>Asset &amp; workforce management</td>
<td>Data governance</td>
</tr>
</tbody>
</table>

**HORIZONTAL**

- Large scale ingest and distribution
- Real-time ELTA (Extract Load Transform Analyze)
- Dimensional computation & aggregation
- Enforcing data quality and data governance requirements
- Real-time data enrichment with reference data
- Real-time machine learning model scoring
Apache Apex

• In-memory, distributed stream processing
  • Application logic broken into components (operators) that execute distributed in a cluster
  • Unobtrusive Java API to express (custom) logic
  • Maintain state and metrics in member variables
  • Windowing, event-time processing

• Scalable, high throughput, low latency
  • Operators can be scaled up or down at runtime according to the load and SLA
  • Dynamic scaling (elasticity), compute locality

• Fault tolerance & correctness
  • Automatically recover from node outages without having to reprocess from beginning
  • State is preserved, checkpointing, incremental recovery
  • End-to-end exactly-once

• Operability
  • System and application metrics, record/visualize data
  • Dynamic changes and resource allocation, elasticity
Native Hadoop Integration

- YARN is the resource manager
- HDFS for storing persistent state
A Stream is a sequence of data tuples

A typical Operator takes one or more input streams, performs computations & emits one or more output streams

- Each Operator is YOUR custom business logic in java, or built-in operator from our open source library
- Operator has many instances that run in parallel and each instance is single-threaded

Directed Acyclic Graph (DAG) is made up of operators and streams
• Operators from library or develop for custom logic
• Connect operators to form application
• Configure operator properties
• Configure scaling and other platform attributes
• Test functionality, performance, iterate
Application Specification

DAG API (compositional)

```java
@ApplicationAnnotation(name = "MyFirstApplication")
public class Application implements StreamingApplication {

    @Override
    public void populateDAG(DAG dag, Configuration conf) {

        LineReader lineReader = dag.addOperator("input", new LineReader());
        Parser parser = dag.addOperator("parser", new Parser());
        UniqueCounter counter = dag.addOperator("counter", new UniqueCounter());
        ConsoleOutputOperator cons = dag.addOperator("console", new ConsoleOutputOperator());
        dag.addStream("lines", lineReader.output, parser.input);
        dag.addStream("words", parser.output, counter.data);
        dag.addStream("counts", counter.count, cons.input);

    }
}
```

Java Stream API (declarative)

```java
ApexStream stream = StreamFactory.fromFolder(localFolder)
    .flatMap((String input) -> Arrays.asList(input.split(" ")))  
    .countByKey().print();
```
Developing Operators

```java
public class Parser extends BaseOperator {

    public transient final DefaultInputPort<String> input = new DefaultInputPort<String>() {
        @Override
        public void process(String s) {
            String[] words = s.split("[\p{Punct}]\s*\"\,"\"\")
            for (String word : words) {
                output.emit(word);
            }
        }
    }

    public transient final DefaultOutputPort<String> output = new DefaultOutputPort<>();
}
```

```java
public class UniqueCounter<K> extends BaseOperator {

    private Map<K, MutableInt> counts = new HashMap<>();

    public transient final DefaultInputPort<K> input = (tuple) -> {
        MutableInt count = counts.get(tuple);
        if (count == null) {
            count = new MutableInt();
            counts.put(tuple, count);
        }
        count.increment();
    }

    @Override
    public void endWindow() {
        for (Map.Entry<K, MutableInt> entry : counts.entrySet()) {
            output.emit(new KeyValPair<K, Integer>(entry.getKey(), entry.getValue().toInteger()));
        }
    }

    public transient final DefaultOutputPort<KeyValPair<K, Integer>> output = new DefaultOutputPort<>();
}
```
### Operator Library

#### Messaging
- Kafka
- JMS (ActiveMQ, ...)
- Kinesis, SQS
- Flume, NiFi

#### NoSQL
- Cassandra, HBase
- Aerospike, Accumulo
- Couchbase/CouchDB
- Redis, MongoDB
- Geode

#### RDBMS
- JDBC
- MySQL
- Oracle
- MemSQL

#### File Systems
- HDFS/Hive
- NFS
- S3

#### Parsers
- XML
- JSON
- CSV
- Avro
- Parquet

#### Transformations
- Filter, Expression, Enrich
- Windowing, Aggregation
- Join
- Dedup

#### Analytics
- Dimensional Aggregations (with state management for historical data + query)

#### Protocols
- HTTP
- FTP
- WebSocket
- MQTT
- SMTP

#### Other
- Elastic Search
- Script (JavaScript, Python, R)
- Solr
- Twitter
**Stateful Processing with Event Time**

**Event Stream**

- k=A, t=4:00
- k=B, t=5:00
- k=B, t=5:59
- k=A, t=4:30

**Processing Time**

- +30s
- +60s
- +90s

**State**

- (All): 1
  - t=4:00: 1
  - k=A, t=4:00: 1

- (All): 4
  - t=4:00: 2
  - t=5:00: 2
  - k=A, t=4:00: 2
  - K=B, t=5:00: 2

- (All): 5
  - t=4:00: 2
  - t=5:00: 3
  - k=A, t=4:00: 2
  - k=A, t=5:00: 1
  - k=B, t=5:00: 2
Windowing - Apache Beam Model

ApexStream<String> stream = StreamFactory
    .fromFolder(localFolder)
    .flatMap(new Split())
    .window(new WindowOption.GlobalWindow(), new
    TriggerOption().withEarlyFiringsAtEvery(Duration.millis(1000)).accumulatingFiredPanes())
    .countByKey(new ConvertToKeyVal()).print();
Fault Tolerance

• Operator state is checkpointed to persistent store
  ◦ Automatically performed by engine, no additional coding needed
  ◦ Asynchronous and distributed
  ◦ In case of failure operators are restarted from checkpoint state

• Automatic detection and recovery of failed containers
  ◦ Heartbeat mechanism
  ◦ YARN process status notification

• Buffering to enable replay of data from recovered point
  ◦ Fast, incremental recovery, spike handling

• Application master state checkpointed
  ◦ Snapshot of physical (and logical) plan
  ◦ Execution layer change log
Checkpointing State

- Distributed, asynchronous
- Periodic callbacks
- No artificial latency
- Pluggable storage
Buffer Server & Recovery

- In-memory PubSub
- Stores results until committed
- Backpressure / spillover to disk
- Ordering, idempotency

Downstream Operators reset

Independent pipelines (can be used for speculative execution)
Recovery Scenario

... EW₂, 1, 3, BW₂, EW₁, 4, 2, 1, BW₁

... EW₂, 1, 3, BW₂, EW₁, 4, 2, 1, BW₁

... EW₂, 1, 3, BW₂, EW₁, 4, 2, 1, BW₁

... EW₂, 1, 3, BW₂, EW₁, 4, 2, 1, BW₁
Processing Guarantees

At-least-once
- On recovery data will be replayed from a previous checkpoint
  - No messages lost
  - Default, suitable for most applications
- Can be used to ensure data is written once to store
  - Transactions with meta information, Rewinding output, Feedback from external entity, Idempotent operations

At-most-once
- On recovery the latest data is made available to operator
  - Useful in use cases where some data loss is acceptable and latest data is sufficient

Exactly-once
- At-least-once + idempotency + transactional mechanisms (operator logic) to achieve end-to-end exactly once behavior
End-to-End Exactly Once

• Important when writing to external systems
• Data should not be duplicated or lost in the external system in case of application failures
• Common external systems
  ◦ Databases
  ◦ Files
  ◦ Message queues
• Exactly-once = at-least-once + idempotency + consistent state
• Data duplication must be avoided when data is replayed from checkpoint
  ◦ Operators implement the logic dependent on the external system
  ◦ Platform provides checkpointing and repeatable windowing
Scalability

Unifier

Logical Diagram

0 ——— 1 ——— 2

Physical Diagram with operator 1 with 3 partitions

0 ➔ 1 ➔ Unifier ➔ 2

NxM Partitions

Logical DAG

0 ——— 1 ——— 2 ——— 3

Physical DAG with (1a, 1b, 1c) and (2a, 2b): Bottleneck on intermediate Unifier

0 ——— 1b ——— Unifier ——— 2b ——— 3

Physical DAG with (1a, 1b, 1c) and (2a, 2b): No bottleneck

0 ——— 1b ——— Unifier ——— 2a ——— 3

APEX

DataTorrent
Advanced Partitioning

**Parallel Partition**

Logical DAG

0 -> 1 -> 2 -> 3 -> 4

Physical DAG

0 -> 1a -> Unifier -> 2 -> 3 -> 4

1b

Physical DAG with Parallel Partition

0 -> 1a -> 2a -> 3a -> Unifier -> 4

1b -> 2b -> 3b

**Cascading Unifiers**

Logical Plan

uopr

Execution Plan, for N = 4; M = 1

uopr1 -> uopr2 -> uopr3 -> uopr4

Execution Plan, for N = 4; M = 1, K = 2 with cascading unifiers

NIC -> Container unifier -> dopr

uopr1 -> uopr2 -> uopr3 -> uopr4

NIC -> Container unifier -> dopr

NIC -> Container unifier -> dopr
Dynamic Partitioning

- Partitioning change while application is running
  - Change number of partitions at runtime based on stats
  - Determine initial number of partitions dynamically
    - Kafka operators scale according to number of kafka partitions
  - Supports re-distribution of state when number of partitions change
  - API for custom scaler or partitioner

*Unifiers not shown*
How dynamic partitioning works

• Partitioning decision (yes/no) by trigger (StatsListener)
  ◦ Pluggable component, can use any system or custom metric
  ◦ Externally driven partitioning example: KafkaInputOperator

• Stateful!
  ◦ Uses checkpointed state
  ◦ Ability to transfer state from old to new partitions (partitioner, customizable)
  ◦ Steps:
    • Call partitioner
    • Modify physical plan, rewrite checkpoints as needed
    • Undeploy old partitions from execution layer
    • Release/request container resources
    • Deploy new partitions (from rewritten checkpoint)
  ◦ No loss of data (buffered)
  ◦ Incremental operation, partitions that don’t change continue processing

• API: Partitioner interface
Compute Locality

- By default operators are deployed in containers (processes) on different nodes across the Hadoop cluster

- Locality options for streams
  - RACK_LOCAL: Data does not traverse network switches
  - NODE_LOCAL: Data transfer via loopback interface, frees up network bandwidth
  - CONTAINER_LOCAL: Data transfer via in memory queues between operators, does not require serialization
  - THREAD_LOCAL: Data passed through call stack, operators share thread

- Host Locality
  - Operators can be deployed on specific hosts

- (Anti-)Affinity
  - Ability to express relative deployment without specifying a host
Performance: Throughput vs. Latency?

High-Throughput and Low-Latency

Apex, Flink w/ 4 Kafka brokers 
2.7 million events/second, Kafka latency limit

Apex w/o Kafka and Redis: 
43 million events/second with more than 90 percent of events processed with the latency less than 0.5 seconds

https://www.datatorrent.com/blog/throughput-latency-and-yahoo/
Recent Additions & Roadmap

- Declarative Java API
- Windowing Semantics following Beam model
- Scalable state management
- SQL support using Apache Calcite
- Apache Beam Runner, SAMOA integration

- Enhanced support for Batch Processing
- Support for Mesos
- Encrypted Streams
- Python support for operator logic and API
- Replacing operator code at runtime
- Dynamic attribute changes
- Named checkpoints
Enterprise Tools

Graphical Application Design and Launch

Data Visualization Dashboard

Management Console

Malhar: Open Source Operators

Streaming Application

Streaming Runtime
High Performance, fault tolerant, Complex & In-Memory Processing of Data In Motion

Hadoop – YARN + HDFS
Certified With All Major Hadoop Distros (Cloudera, Hortonwoks, Pivotal, MapR)

Physical | Virtual | Cloud
Monitoring Console

Logical View

Physical View
Real-Time Dashboards
Maximize Revenue w/ real-time insights

PubMatic is the leading marketing automation software company for publishers. Through real-time analytics, yield management, and workflow automation, PubMatic enables publishers to make smarter inventory decisions and improve revenue performance.

**Business Need**

- Ingest and analyze high volume clicks & views in real-time to help customers improve revenue
  - 200K events/second data flow
- Report critical metrics for campaign monetization from auction and client logs
  - 22 TB/day data generated
- Handle ever increasing traffic with efficient resource utilization
- Always-on ad network

**Apex based Solution**

- DataTorrent Enterprise platform, powered by Apache Apex
- In-memory stream processing
- Comprehensive library of pre-built operators including connectors
- Built-in fault tolerance
- Dynamically scalable
- Real-time query from in-memory state
- Management UI & Data Visualization console

**Client Outcome**

- Helps PubMatic deliver ad performance insights to publishers and advertisers in real-time instead of 5+ hours
- Helps Publishers visualize campaign performance and adjust ad inventory in real-time to maximize their revenue
- Enables PubMatic reduce OPEX with efficient compute resource utilization
- Built-in fault tolerance ensures customers can always access ad network
GE is dedicated to providing advanced IoT analytics solutions to thousands of customers who are using their devices and sensors across different verticals. GE has built a sophisticated analytics platform, Predix, to help its customers develop and execute Industrial IoT applications and gain real-time insights as well as actions.

<table>
<thead>
<tr>
<th>Business Need</th>
<th>Apex based Solution</th>
<th>Client Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Ingest and analyze high-volume, high speed data from thousands of devices, sensors per customer in real-time without data loss</td>
<td>• Ingestion application using DataTorrent Enterprise platform</td>
<td>• Helps GE improve performance and lower cost by enabling real-time Big Data analytics</td>
</tr>
<tr>
<td>• Predictive analytics to reduce costly maintenance and improve customer service</td>
<td>• Powered by Apache Apex</td>
<td>• Helps GE detect possible failures and minimize unplanned downtimes with centralized management &amp; monitoring of devices</td>
</tr>
<tr>
<td>• Unified monitoring of all connected sensors and devices to minimize disruptions</td>
<td>• In-memory stream processing</td>
<td>• Enables faster innovation with short application development cycle</td>
</tr>
<tr>
<td>• Fast application development cycle</td>
<td>• Built-in fault tolerance</td>
<td>• No data loss and 24x7 availability of applications</td>
</tr>
<tr>
<td>• High scalability to meet changing business and application workloads</td>
<td>• Dynamic scalability</td>
<td>• Helps GE adjust to scalability needs with auto-scaling</td>
</tr>
</tbody>
</table>
Silver Spring Networks helps global utilities and cities connect, optimize, and manage smart energy and smart city infrastructure. Silver Spring Networks receives data from over 22 million connected devices, conducts 2 million remote operations per year.

### Business Need
- Ingest high-volume, high speed data from millions of devices & sensors in real-time without data loss
- Make data accessible to applications without delay to improve customer service
- Capture & analyze historical data to understand & improve grid operations
- Reduce the cost, time, and pain of integrating with 3rd party apps
- Centralized management of software & operations

### Apex based Solution
- DataTorrent Enterprise platform, powered by Apache Apex
  - In-memory stream processing
  - Pre-built operators/ connectors
  - Built-in fault tolerance
  - Dynamically scalable
  - Management UI console

### Client Outcome
- Helps Silver Spring Networks ingest & analyze data in real-time for effective load management & customer service
- Helps Silver Spring Networks detect possible failures and reduce outages with centralized management & monitoring of devices
- Enables fast application development for faster time to market
- Helps Silver Spring Networks scale with easy to partition operators
- Automatic recovery from failures
Who is using Apex?

• **Powered by Apex**
  - Also using Apex? Let us know to be added: users@apex.apache.org or @ApacheApex

• **Pubmatic**
  - [https://www.youtube.com/watch?v=JSXpgfQFcU8](https://www.youtube.com/watch?v=JSXpgfQFcU8)

• **GE**
  - [https://www.youtube.com/watch?v=hmaSkXhHnu0](https://www.youtube.com/watch?v=hmaSkXhHnu0)

• **SilverSpring Networks**
  - [https://www.youtube.com/watch?v=8VORISKeSjl](https://www.youtube.com/watch?v=8VORISKeSjl)
Q&A
Resources

• http://apex.apache.org/
• Learn more - http://apex.apache.org/docs.html
• Subscribe - http://apex.apache.org/community.html
• Download - http://apex.apache.org/downloads.html
• Follow @ApacheApex - https://twitter.com/apacheapex
• Meetups - https://www.meetup.com/topics/apache-apex/
• Examples - https://github.com/DataTorrent/examples
• Slideshare - http://www.slideshare.net/ApacheApex/presentations
• https://www.youtube.com/results?search_query=apache+apex
• Free Enterprise License for Startups - https://www.datatorrent.com/product/startup-accelerator/