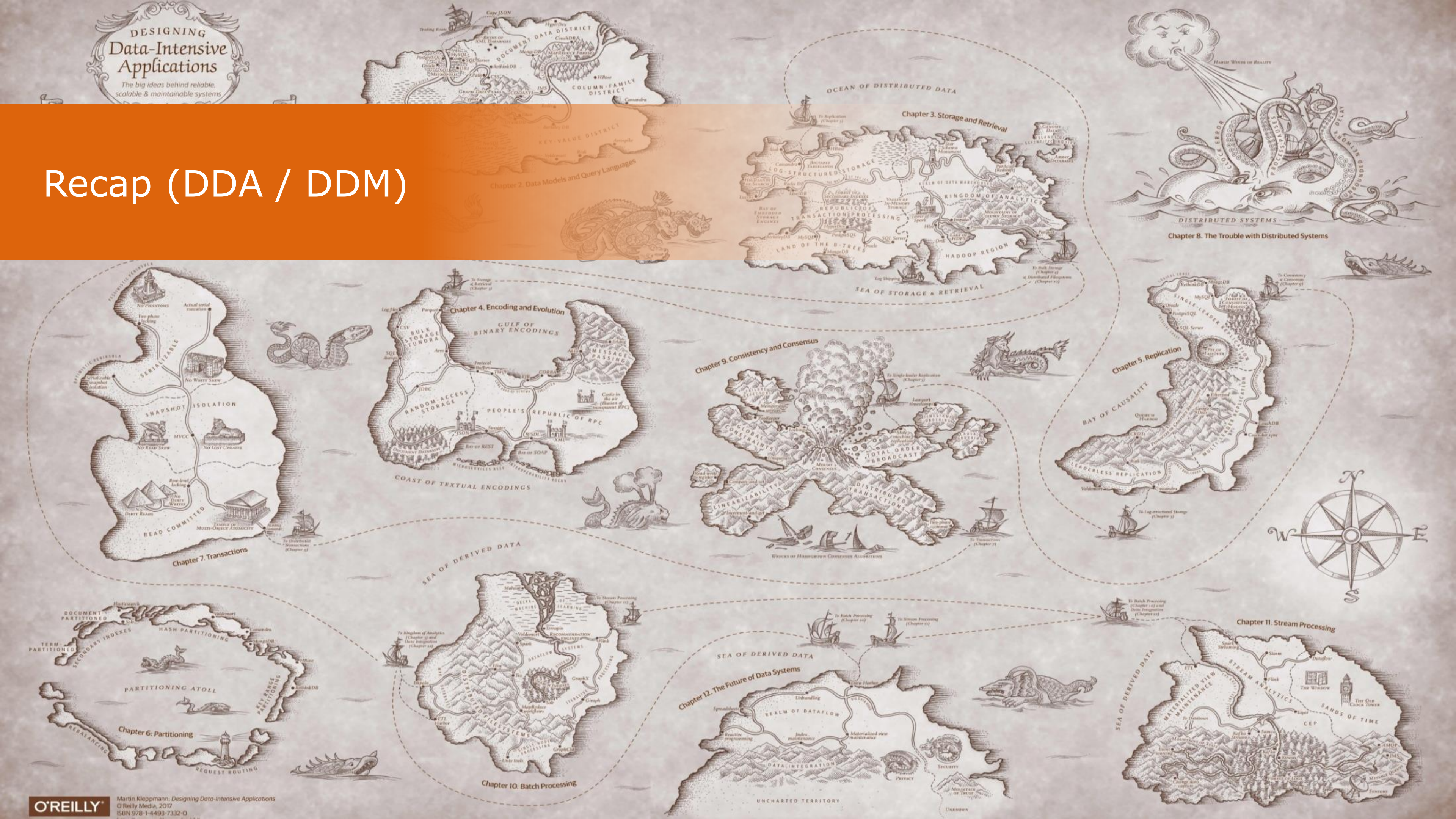
Seminar

Mining Streaming Data

Alexander Albrecht
bakdata
Thorsten Papenbrock
Hasso Plattner Institut

Recap (DDA / DDM)



Streams

Data Stream

- Any data that is incrementally made available over time
- Examples:
  - Unix `stdin` and `stdout`
  - Filesystem APIs (e.g. Java's `FileInputStream`)
  - Online media delivery (audio/video streaming)
- Creation from …
  - static data: files or databases (read records line-wise)
  - dynamic data: sensor readings, service calls, transmitted data, logs, …

Event

- = an immutable record in a stream (often with timestamp)
- "Something that happened"
- Encoded in Json, XML, CSV, … maybe in binary format

Any format that allows incremental appends

Types of Systems

Services (online systems)

- Accept requests and send responses
- Performance measure: response time and availability
- Expected runtime: milliseconds to seconds

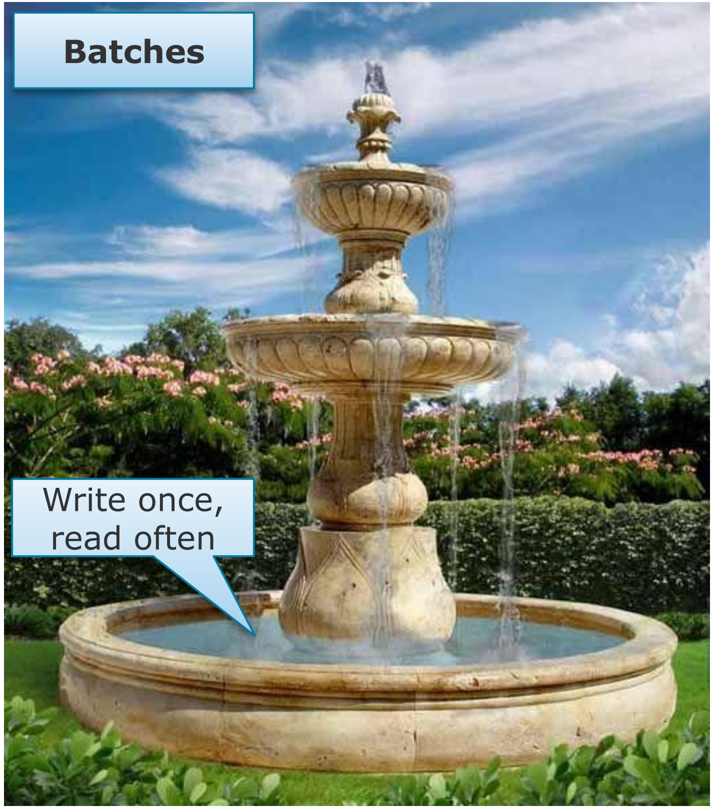
Batch processing systems (offline systems)

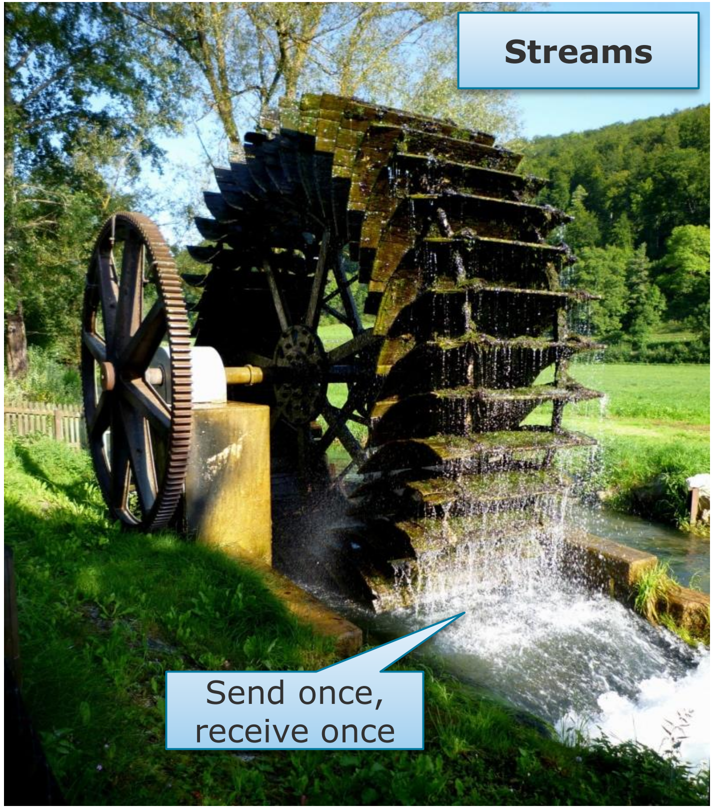
- Take (large amounts of) data; run (complex) jobs; produce some output
- Performance measure: throughput (i.e., data per time)
- Expected runtime: minutes to days

Stream processing systems (near-real-time systems)

- Consume volatile inputs; operate stream jobs; produce some output
- Performance measure: throughput and precision
- Expected runtime: near-real-time (i.e., as data arrives)

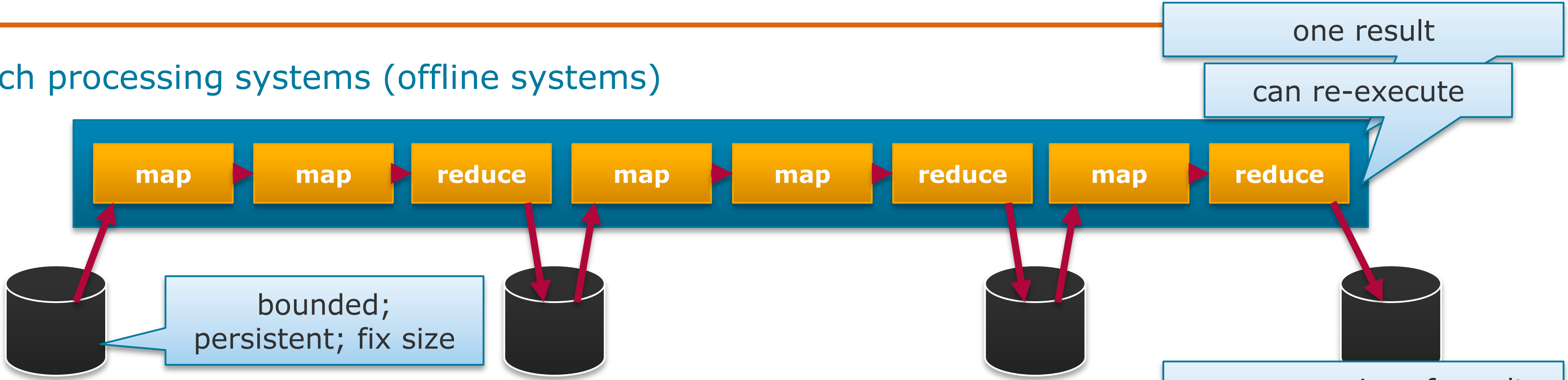
Batch vs. Stream



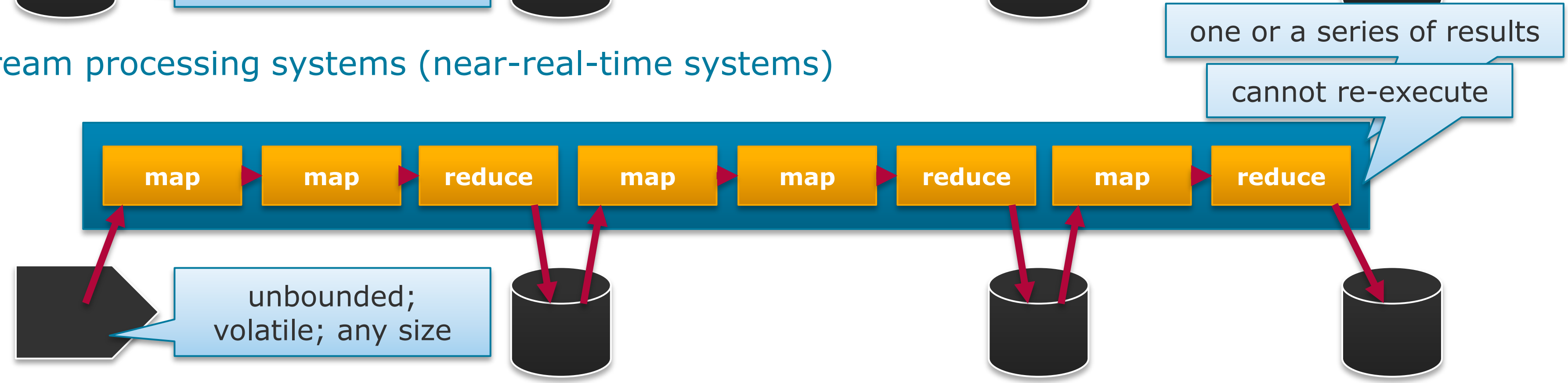


Types of Systems

Batch processing systems (offline systems)



Stream processing systems (near-real-time systems)



Use Cases for Streaming Data

Sensor Processing

- Continuous and endless readings by nature

Process Monitoring

- Side effects of processes that are continuously observed

Location Tracking

- Continuous location updates of certain devices

Log Analysis

- Digital footprints of applications that grow continuously

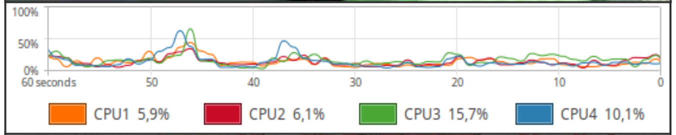
User Interaction

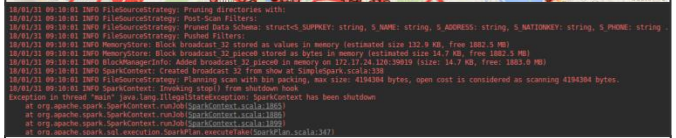
- Continuous and oftentimes bursty click- and call-events

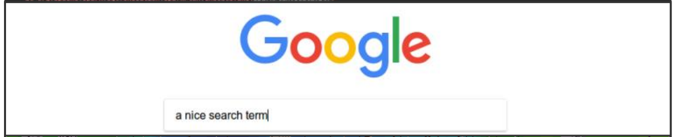
Market and Climate Prediction

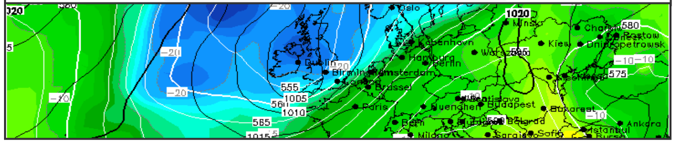
- Changing stock market prices and weather characteristics

…









Transmitting Event Streams

Message Brokers: Persist or Forget

[1]Java Message Service (JMS) 2.0 Specification
[2]Advanced Message Queuing Protocol (AMQP) Specification

Persist ⟷ Forget

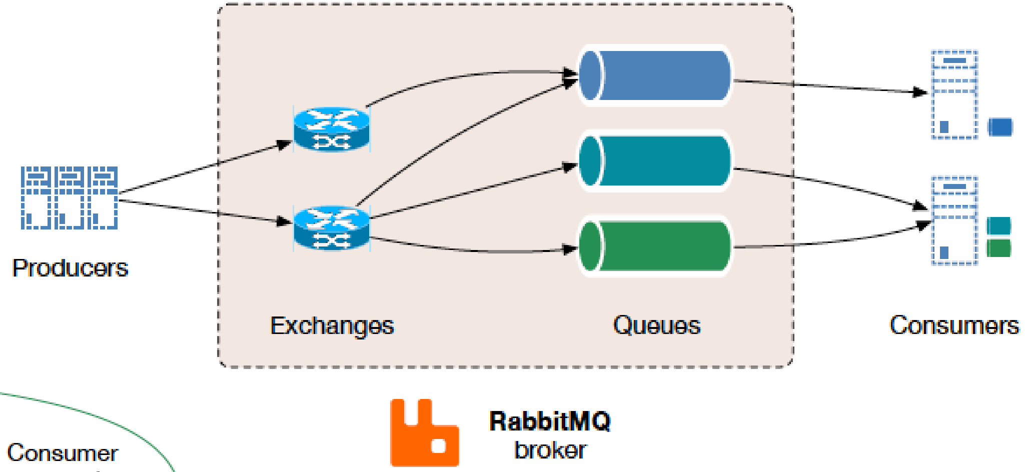
- Keep all queue content
  (until reaching size or time limit)
- No need to track consumers
- Let consumers go back in time
  - Database-like
- Log-based Message Broker
  (e.g. Kafka, Kinesis or DistributedLog)

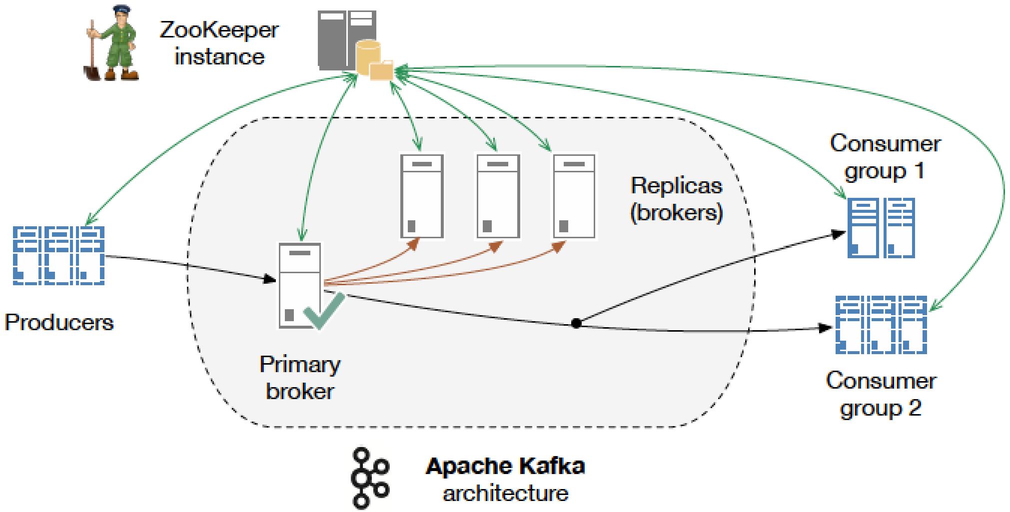
- Remove processed queue content
  (immediately after acknowledgement)
- Track consumers to forget old content
- The past is past
  - Volatile, light-weight
- JMS[1] or AMQP[2] Message Brokers
  (e.g. RabbitMQ, ActiveMQ or HornetQ)

Message Brokers: Persist or Forget

https://content.pivotal.io/blog/understanding-when-to-use-rabbitmq-or-apache-kafka

http://kth.diva-portal.org/smash/get/diva2:813137/FULLTEXT01.pdf





Message Brokers: Persist or Forget

[1]Java Message Service (JMS) 2.0 Specification
[2]Advanced Message Queuing Protocol (AMQP) Specification

Persist ⟷ Forget

Persist

Keep all queue content
(until reaching size or time limit)

No need to track consumers

Let consumers go back in time

- ➢ Database-like

Log-based Message Broker
(e.g. Kafka, Kinesis or DistributedLog)

kafka

Forget

Remove processed queue content
(immediately after acknowledgement)

Track consumers to forget old content

The past is past

- ➢ Volatile, light-weight

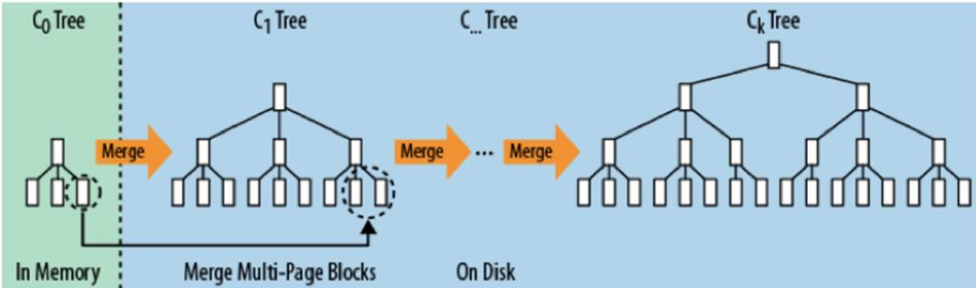
JMS[1] or AMQP[2] Message Brokers
(e.g. RabbitMQ, ActiveMQ or HornetQ)

RabbitMQ

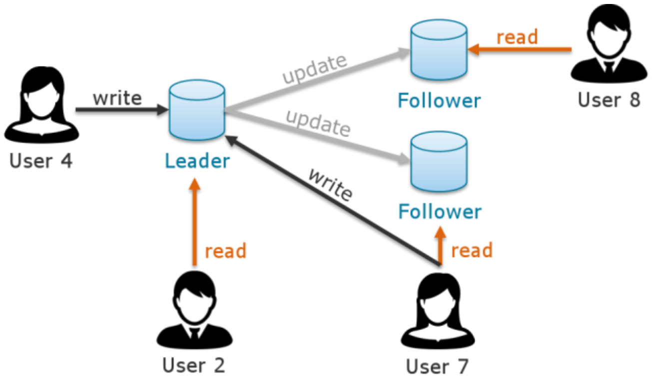
Log-based Massage Broker

Partitioned Logs

- Message Broker that persist queues as logs on disk (distributed, replicated)
- Recall …
  - LSM-Trees with B-Trees and SSTables



  - Leader-based replication



Log-based Massage Broker

Topics and Partitions

- Topics are logical groupings for event streams
  - e.g. click-events, temperature-readings, location-signals
  - Every topic is created with a fixed number of partitions
- Partitions are ordered lists of logically dependent events in a topic
  - e.g. click-events by user, temperature-readings by sensor, location-signals by car
  - Provide "happens-before semantic" for these events
  - Order is valid within each partition, not across different partitions
  - Are accessed sequentially
    - Producers write new events sequentially
    - Consumers read events sequentially
  - Purpose:
    - Parallelism: to read a topic in parallel
    - Load-balancing: to store the events of one topic on multiple nodes

In many cases, event ordering is not a concern and partitions are simply arbitrary splits of a topic (for parallelization and load-balancing)

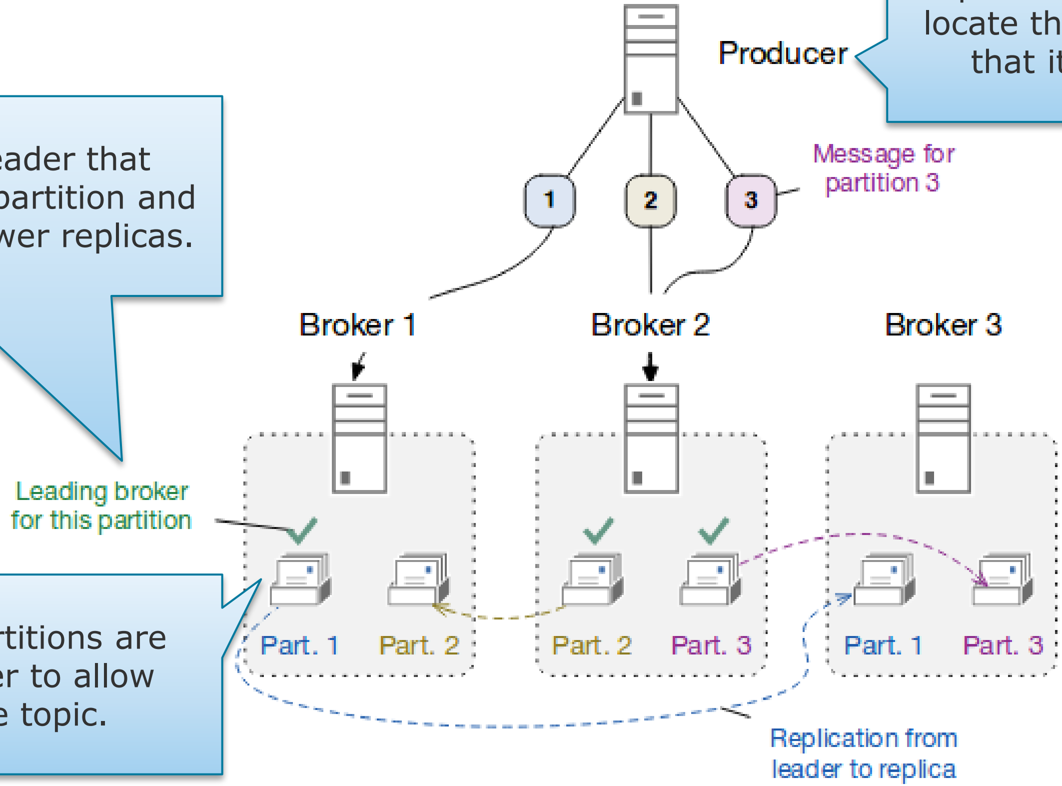
Log-based Massage Broker

Topics and Partitions

A producer uses ZooKeeper to locate the leader of a partition that it wants to write to.

Every partition has a leader that accepts all writes to that partition and forwards them to its follower replicas.

Leaders for different partitions are distributed in the cluster to allow parallel writes to one topic.



Log-based Massage Broker

Producers and Consumers

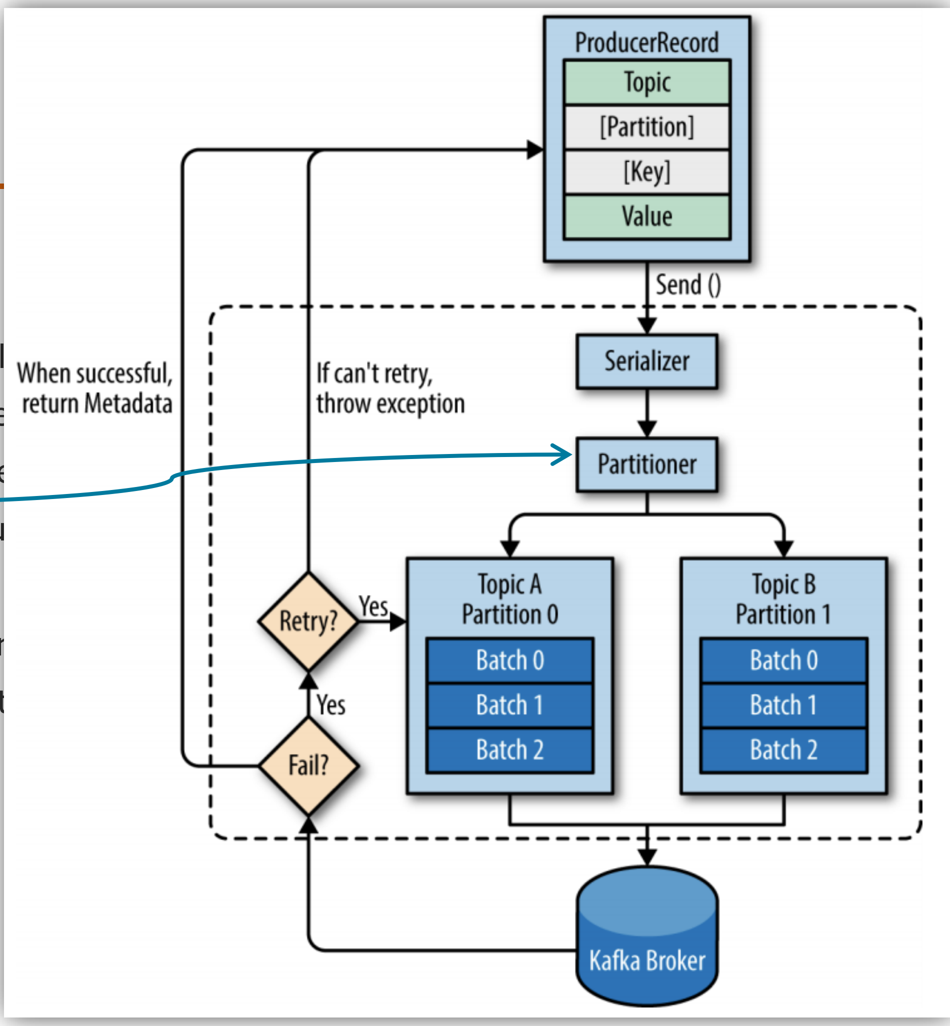
- Producers
  - Post to concrete partitions within a topic (only one leader can take these posts)
  - Define a Partitioner-strategy (on the producer side) to decide which partition is next
    - Round-Robin Partitioner-strategy is used by default
    - Custom Partitioner-strategies let producers define semantic grouping functions
- Consumers
  - Read concrete partitions within a topic (all broker with that partition can take these reads)
  - Hold an offset pointer for every partition that they read (on consumer side)
  - Poll and wait (no callback registration)

> "Kafka does not track acknowledgments from consumers […]. Instead, it *allows* consumers to use Kafka to track their position (offset) in each partition."
>
> (Book: Kafka - The Definite Guide)

Log-based Massage Broker

Producers and Consumers

- Producers
  - Post to concrete partitions within a topic (onl
  - Define a Partitioner-strategy (on the produce
    - Round-Robin Partitioner-strategy is use
    - Custom Partitioner-strategies let produ
- Consumers
  - Read concrete partitions within a topic (all br
  - Hold an offset pointer for every partition that
  - Poll and wait (no callback registration)

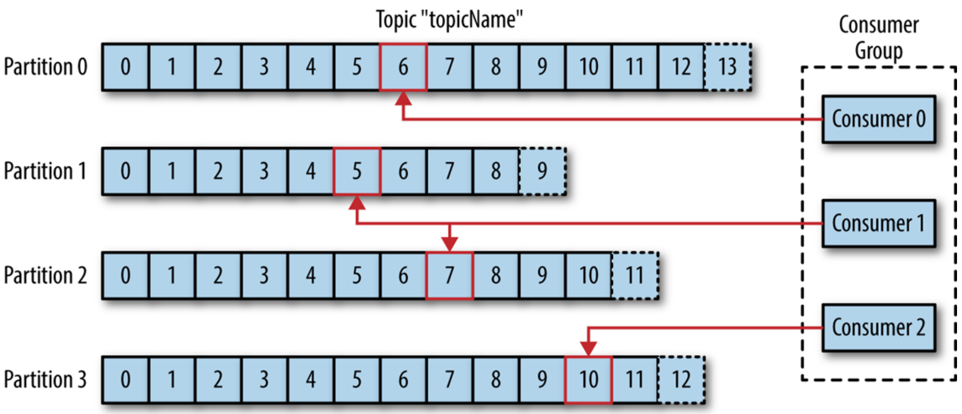


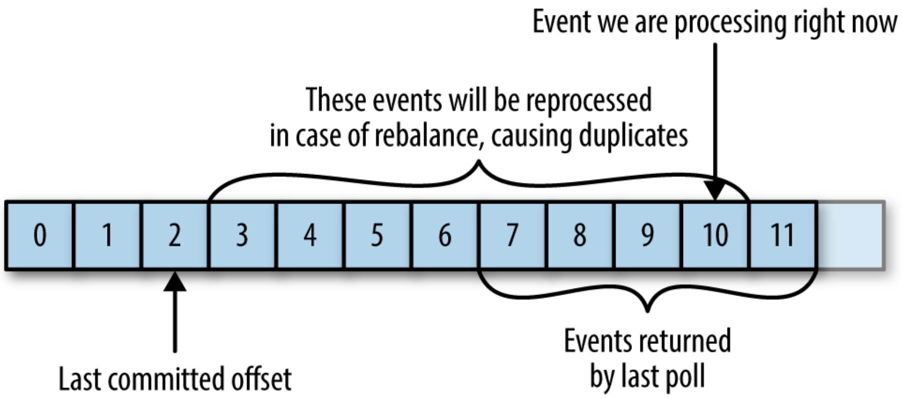
Log-based Massage Broker

Producers and Consumers

- Consumer Groups
  - A group of consumers that processes all events of one topic in parallel
  - The offsets for a consumer group can be managed by Kafka on server side
    - A dedicated group coordinator manages offsets, membership, scheduling etc.
    - Consumer commit successfully processed offsets to the group coordinator so that the coordinator can re-assign partitions to consumers

And in this way, Kafka kind of knows its consumers …



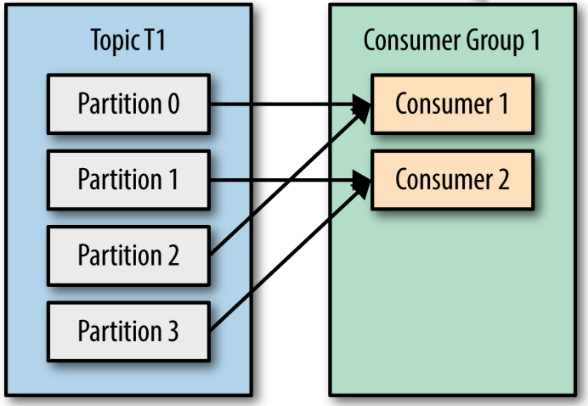


Log-based Massage Broker

Producers and Consumers

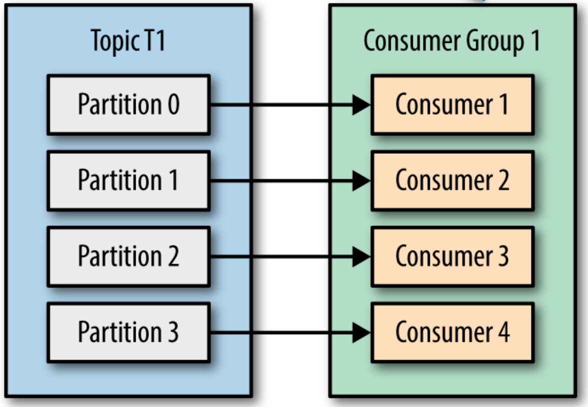
#partitions > #consumer

- Consumer take multiple partitions and process them alternatingly



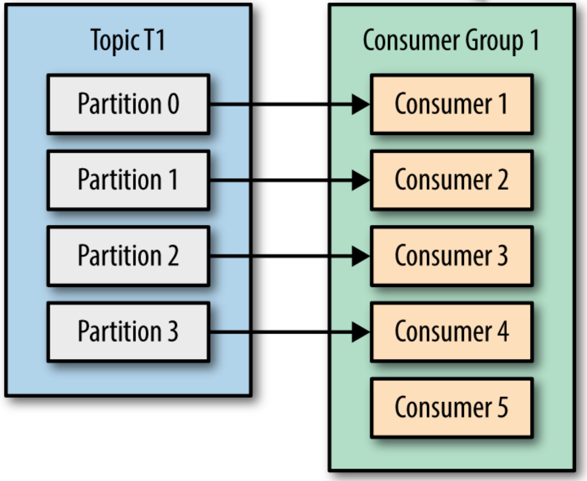
#partitions = #consumer

- Every consumer takes one partition; maximum parallelism



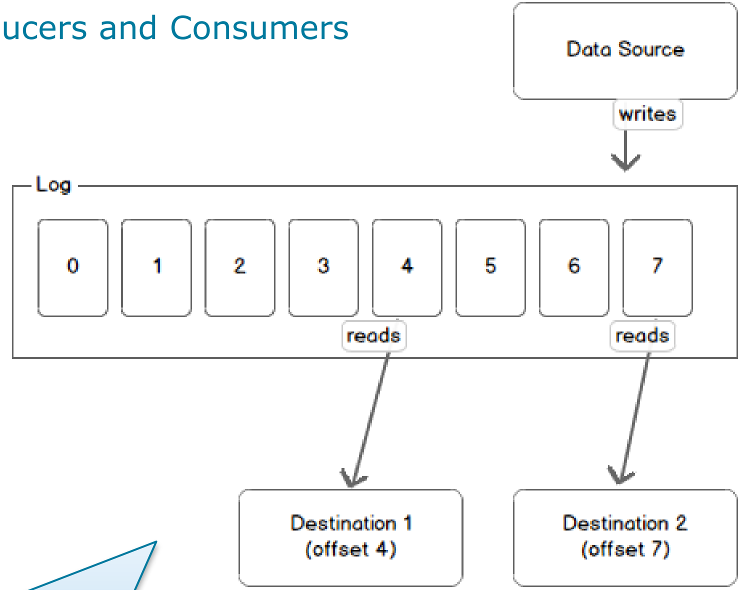
#partitions < #consumer

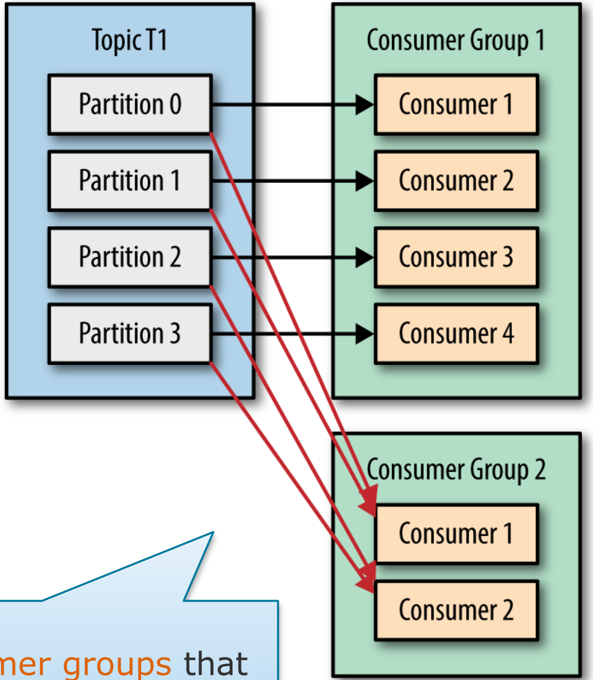
- Some consumers idle, because the group reads every partition exactly once



Log-based Massage Broker

Producers and Consumers





Different consumers that read the same partition in parallel and at different locations.

Different consumer groups that read same partitions in parallel (and at different locations).

Log-based Massage Broker

Kafka APIs

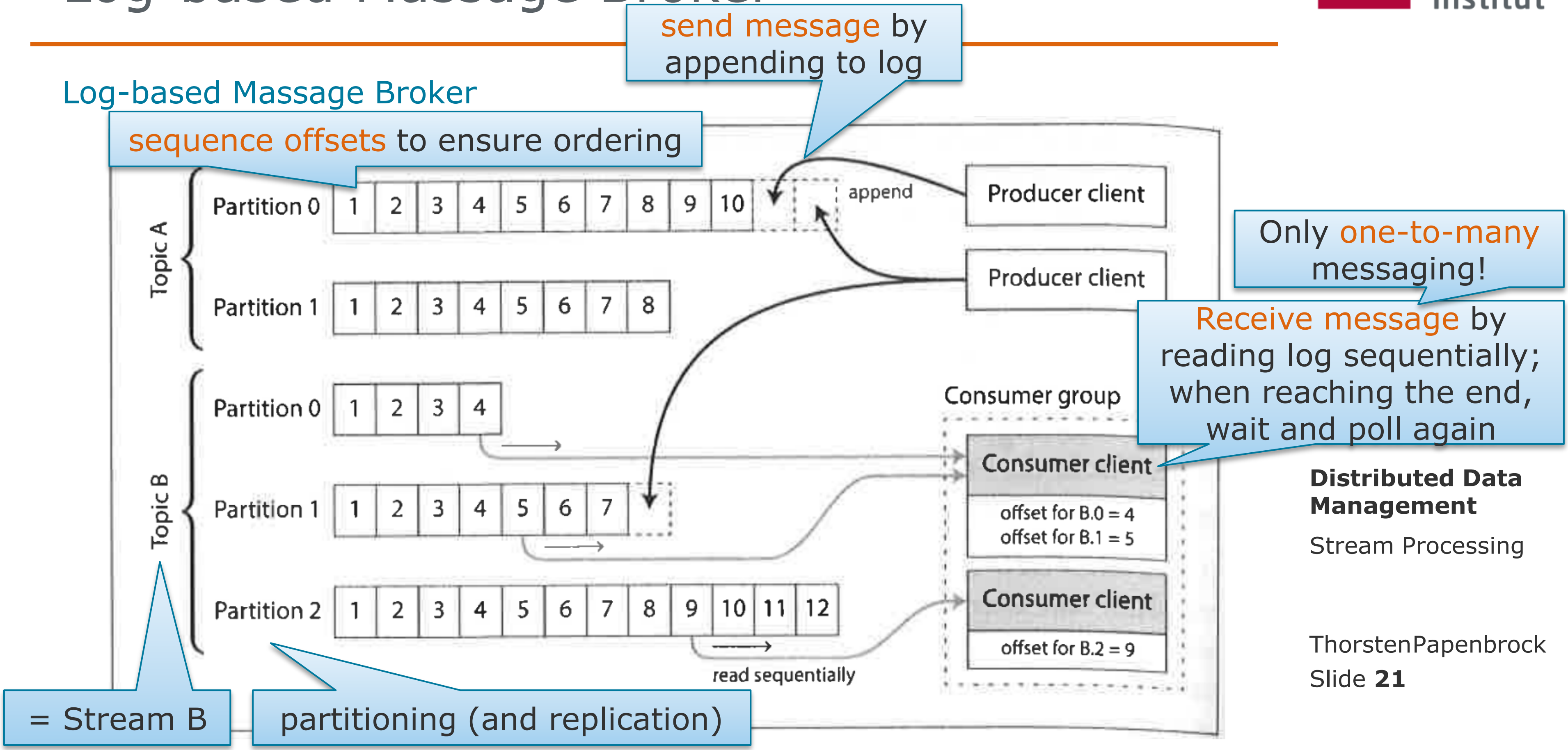
- Communication with Kafka happens via a specific APIs
- The API can manage the specifics of the reading/writing process transparently
  - e.g. offset-tracking (consumers) and partition-scheduling (producers)
- Two options:
  - A rich API that offers high abstraction, but limited control functions.
  - A low-level API that provides access to offsets and allows consumers to rewind them as the need.

Event lifetime

- Configurable:
  - By time of event
  - Max partition size

Log-based Massage Broker

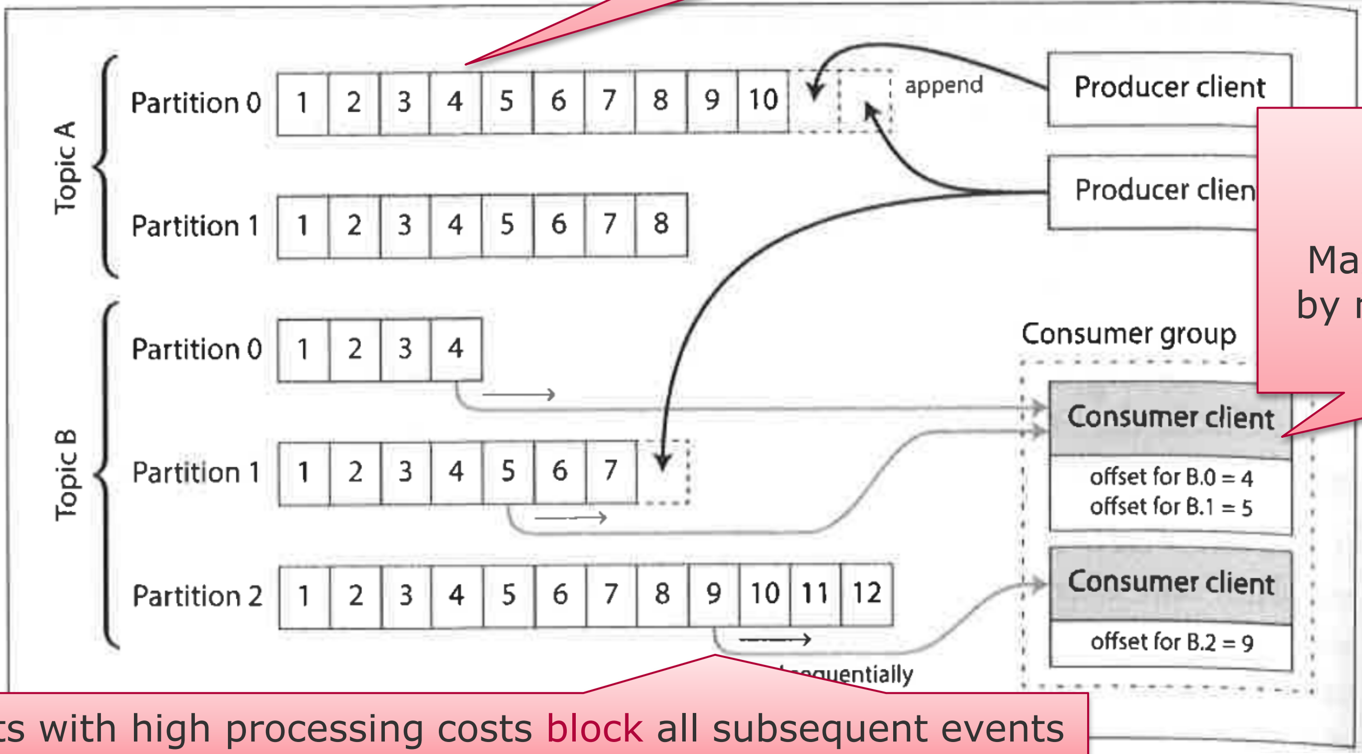
Log-based Massage Broker



Transmitting Event Streams

Log-based Massage Broker

Log-based Massage Broker



Storing a history for events costs memory

Example:

6 TB of disk capacity (= log size)
150 MB/s write throughput

11 h until an event is forgotten
(at maximum event throughput!)

No one-to-one scheduling:

Max parallelism bound by number of partitions in a topic!

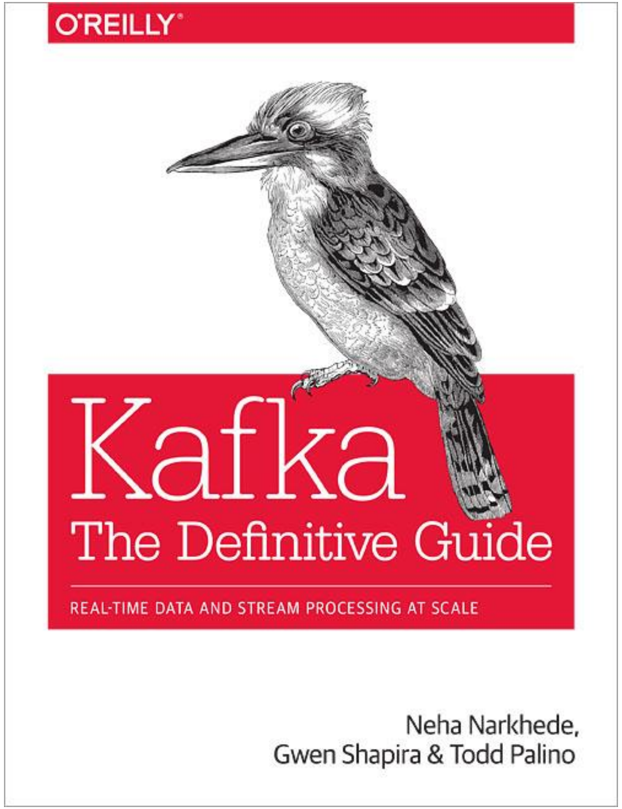
Events with high processing costs block all subsequent events

Log-based Massage Broker

Further reading

- Kafka: The Definitive Guide

- https://www.oreilly.com/library/view/kafka-the-definitive/9781491936153/



Message Brokers: Persist or Forget

[1]Java Message Service (JMS) 2.0 Specification
[2]Advanced Message Queuing Protocol (AMQP) Specification

- Keep all queue content
  (until reaching size or time limit)
- No need to track consumers
- Let consumers go back in time
  - ➢ Database-like
- Log-based Message Broker
  (e.g. Kafka, Kinesis or DistributedLog)

- Remove processed queue content
  (immediately after acknowledgement)
- Track consumers to forget old content
- The past is past
  - ➢ Volatile, light-weight
- JMS[1] or AMQP[2] Message Brokers
  (e.g. RabbitMQ, ActiveMQ or HornetQ)

Use if **throughput** matters,
event processing costs are similar and
the **order of messages** is important

Use if **one-to-one scheduling** is needed,
event processing costs differ and
the order of messages is insignificant

Message Brokers: Persist or Forget

[1]Java Message Service (JMS) 2.0 Specification
[2]Advanced Message Queuing Protocol (AMQP) Specification

Persist ⟷ Forget

- Keep all queue content
  (until reaching size or time limit)
- No need to track consumers
- Let consumers go back in time
  - Database-like
- Log-based Message Broker
  (e.g. Kafka, Kinesis or DistributedLog)

Use if **throughput** matters, event processing costs are similar and the **order of messages** is important

Wait **throughput**?

Yes, because …

- dumping events to storage instead of routing them to consumers is faster
- broker does not need to track acknowledgements for every event (only consumers track their queue offset)
- broker can utilize batching and pipelining internally

Scenarios

Complex Event Processing (CEP)

- "Check a stream for patterns; whenever something special happens, raise a flag"
- Similar to pattern matching with regular expressions (often SQL-dialects)
- Implementations: Esper, IBM InfoSphere, Apama, TIBICO StreamBase, SQLstream

Stream Analytics

- "Transform or aggregate a stream; continuously output current results"
- Often uses statistical metrics and probabilistic algorithms:
  - Bloom filters (set membership)
  - HyperLogLog (cardinality estimation)
  - HDHistogram, t-digest, decay (percentile approximation)
- Implementations: Storm, Flink, Spark Streaming, Concord, Samza, **Kafka Streams**, Google Cloud Dataflow, Azure Stream Analytics

Bounded memory consumption

Approximation is often used for optimization, but Stream Processing is **not** inherently approximate!

Scenarios

Stream = Database (using log compaction etc.)

Usually consider entire stream, i.e., no window!

Maintaining Materialized Views

- "Serve materialized views with up-to-date data from a stream"
- Views are also caches, search indexes, data warehouses, and any derived data system
- Implementations: Samza, **Kafka Streams** (but also works with Flink, Spark, and co.)

Search on Streams

- "Search for events in the stream; emit any event that matches the query"
- Similar to CEP but the standing queries are indexed, less complex, and more in number
- Implementations: Elasticsearch

Message Passing

- "Use the stream for event communication; actors/processes consume and produce events"
- Requires non-blocking one-to-many communication
- Implementations: Any message broker; RPC systems with one-to-many support

Challenges and Limits

Goal

- Query and analyze streaming data in real-time (i.e. as data passes by)

Challenges

- Limited memory resources (but endlessly large volumes of data)
  - Only a fixed-size window of the stream is accessible at a time
- Old data is permanently gone (and not accessible any more)
  - Only one-pass algorithms can be used
- Endlessness contradicts certain operations
  - E.g. sorting makes no sense, i.e., no sort-merge-joins or –groupings (on the entire stream!)
- Input cannot be re-read or easily back-traced
  - Fault tolerance must be ensured differently

Mining Streaming Data

Learning Goals

a) Understand, implement, and deploy a challenging research algorithm.
(no optimization required)

b) Learn about state-of-the-art streaming techniques.

c) Build an algorithm for data streams using Kafka and Kafka Streams.

d) Solve problems that arise from distributed computing.

e) Evaluate the quality and performance of your algorithm.

f) Write a scientific documentation.

g) Reveal new research questions for distributed computing (at best).

Prerequisites

- Database knowledge (ideally Database System I and Database Systems II)
- Data streaming and distributed programming knowledge
(ideally Distributed Data Analytics or Distributed Data Management)

Organization

Hasso Plattner Institut

Tasks: From Paper to Production

1) Choose a paper.
2) Study the literature of your topic (books, papers, and online material).
3) Design a distributed algorithm with Kafka Streams that solves the problem of your paper.
4) Evaluate your solution w.r.t. accuracy/quality and performance.
5) Document your approach by writing a scientific documentation about as a GitHub page.

Grading

- 10% Active participation during all seminar events.
- 00% Regular meetings with advisor.
- 10% Short presentation of the selected research paper.
- 15% Intermediate presentation demonstrating insights regarding your research prototype.
- 15% Final presentation demonstrating your solution.
- 20% Implementation of a research prototype with Kafka and Kafka Streams (on GitHub).
- 30% Documentation (on GitHub).

Organization

Metadata

- Extent: 4 SWS
- Location: Campus II, Building F, Room F-2-10
- Dates: Wednesdays, 11 - 12:30 PM
- Class: At most 8 participants (4 teams á 2 students)
- Register: Informal email to thorsten.papenbrock@hpi.de by April 12 (notification April 15)

Registration Email

- Add your distributed programming experience (e.g. DDA, DDM, some other course, or project).
- Add a ranking of up to three papers that interest you (from the list shown today or own suggestions).
  - We do the final paper assignment in our first Kick-off meeting; so this is not a commit!
- <optional> Add a team partner; you get either accepted or rejected together if seats get tight.

Organization

Small team meetings

- Regular meetings with supervisor (Alexander or Thorsten)

Schedule (tentative)

- April 12: (Email) Registration
- April 15: (Email) Notification
- April 17: Kick-off: Paper Selection & Team Building
- April 24: Guest Speaker Michael Noll (Confluent): "Kafka in Theory and Practice“
- May 1: -
- May 8: Guest Speaker Arvid Heise (bakdata): "Kafka Streams with Q&A"
- May 15: First Presentations: Paper & Implementation Approach

Project duration

- Intermediate presentation: ~5. June
- Final presentation: ~10. July

Paper Suggestions

- Clustering Stream Data by Exploring the Evolution of Density Mountain
  Shufeng Gong, Yanfeng Zhang, and Ge Yu, VLDB 2017.
- Scalable Kernel Density Estimation-based Local OutlierDetection over Large Data Streams
  Xiao Qin, Lei Cao, Elke A. Rundensteiner, and Samuel Madden, EDBT 2019.
- Extremely Fast Decision Tree
  Chaitanya Manapragada, Geoffrey I. Webb, and Mahsa Salehi, KDD 2018.
- Sketching Linear Classifiers over Data Streams
  Kai Sheng Tai, Vatsal Sharan, Peter Bailis, and Gregory Valiant, SIGMOD 2018.
- Cold Filter: A Meta-Framework for Faster and More Accurate Stream Processing
  Yang Zhou, Tong Yang, Jie Jiang, Bin Cui, Minlan Yu, Xiaoming Li, and Steve Uhlig, SIGMOD 2018.
- GraphJet: Real-Time Content Recommendations at Twitter
  Aneesh Sharma, Jerry Jiang, Praveen Bommannavar, Brian Larso, and Jimmy Lin, VLDB 2016.
- Online Social Media Recommendation over Streams
  Xiangmin Zhou, Dong Qin, Xiaolu Lu, Lei Chen, and Yanchun Zhang, ICDE 2019.
- SpotLight: Detecting Anomalies in Streaming Graphs
  Dhivya Eswaran, Christos Faloutsos, Sudipto Guha, and Nina Mishra, KDD 2018.

Paper Suggestions

Clustering Stream Data by Exploring the Evolution of Density Mountain

Shufeng Gong[1,3], Yanfeng Zhang[1,3], Ge Yu[1,2,3]
[1]Northeastern University, [2]Liaoning University
[3]Key laboratory of Medical Image Computing (Northeastern University), Ministry of Education
Shenyang, China
gongsf@stumail.neu.edu.cn, {zhangyf, yuge}@mail.neu.edu.cn

ABSTRACT

Stream clustering is a fundamental problem in many streaming data analysis applications. Comparing to classical batch-mode clustering, there are two key challenges in stream clustering: (i) Given that input data are changing continu- ously, how to incrementally up efficiently? (ii) Given that clust the evolution of data, how to c activities? Unfortunately, most algorithms can neither update t nor track the evolution of cluste

In this paper, we propose a s *EDMStream* by exploring the Ev The *density mountain* is used tribution, the changes of which evolution. We track the evolutio the changes of density mount efficient data structures and f that the update of density mou makes online clustering possible on synthetic and real datasets the state-of-the-art stream clus Stream, DenStream, DBSTRE algorithm is able to response faster (say 7-15x faster than t and at the same time achieve Furthermore, *EDMStream* succ evolution activities.

PVLDB Reference Format:
Shufeng Gong, Yanfeng Zhang, Ge by Exploring the Evolution of Dens 393-405, 2017.
DOI: 10.1145/3164135.3164136

1. INTRODUCTION

Recent advances in both hardware and software have

over time are referred to as data streams [2]. Clustering stream data is one of the most fundamental problems in many streaming data analysis applications. Basically, it groups streaming data on the basis of their similarity, where data evolves over time and arrives in an unbounded stream.

For the first challenge, most existing solutions [5, 7] summarize data points in stream using summary structures (e.g., micro-clusters [3, 5], grids[7]) and update these sum-

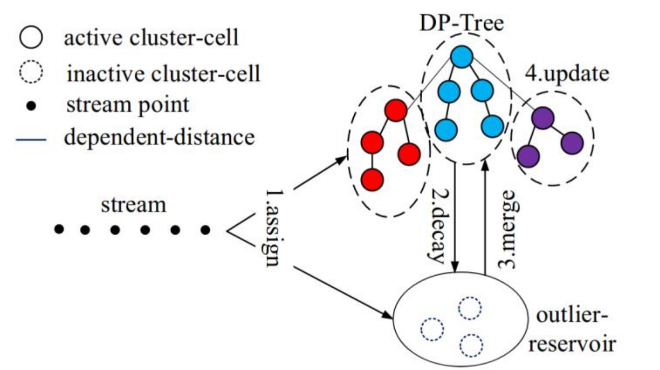


Figure 5: *EDMStream* Overview.

Problem

- Efficient and dynamic clustering of multi-dimensional stream data

Solution

- EDMStream, an algorithm that continuously updates the clusters (described by “Density Mountains”) with newly arriving stream data

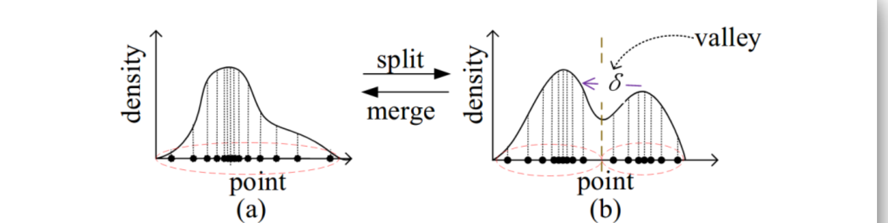
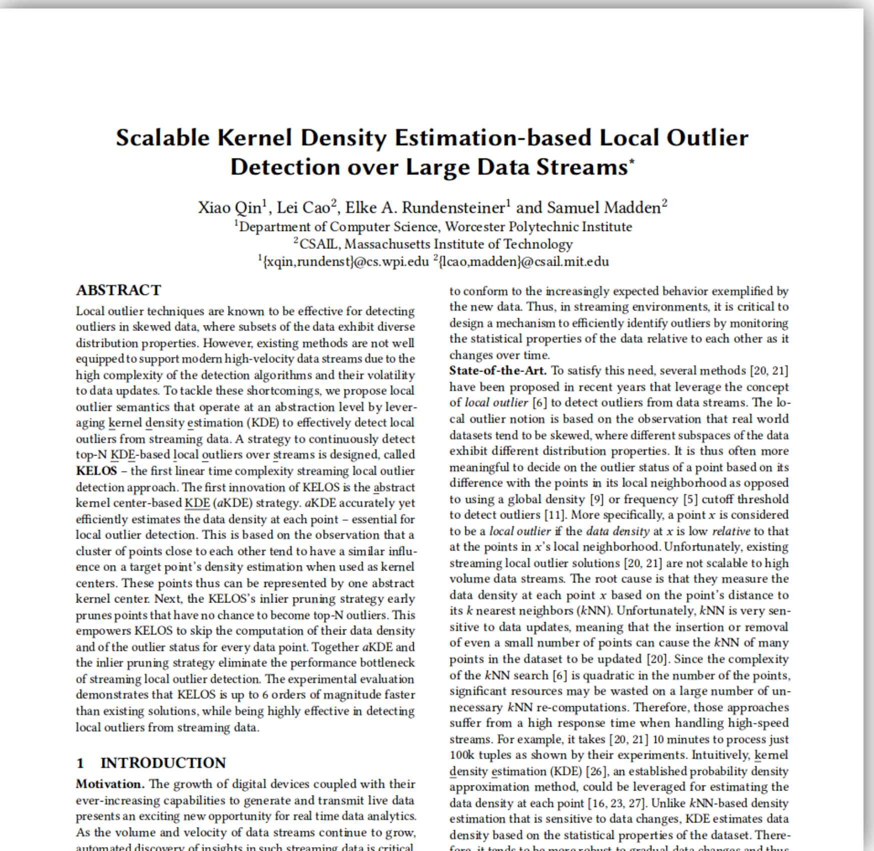


Figure 1: The shape of density mountain changes as the (1-dimension) data distribution evolves.

Mining Streaming Data
Paper Suggestions



Scalable Kernel Density Estimation-based Local Outlier Detection over Large Data Streams*

Xiao Qin[1], Lei Cao[2], Elke A. Rundensteiner[1] and Samuel Madden[2]
[1]Department of Computer Science, Worcester Polytechnic Institute
[2]CSAIL, Massachusetts Institute of Technology
[1]{xqin,rundenst}@cs.wpi.edu [2]{lcao,madden}@csail.mit.edu

ABSTRACT

Local outlier techniques are known to be effective for detecting outliers in skewed data, where subsets of the data exhibit diverse distribution properties. However, existing methods are not well equipped to support modern high-velocity data streams due to the high complexity of the detection algorithms and their volatility to data updates. To tackle these shortcomings, we propose local outlier semantics that operate at an abstraction level by leveraging kernel density estimation (KDE) to effectively detect local outliers from streaming data. A strategy to continuously detect top-N KDE-based local outliers over streams is designed, called **KELOS** – the first linear time complexity streaming local outlier detection approach. The first innovation of KELOS is the abstract kernel center-based KDE (*a*KDE) strategy. *a*KDE accurately yet efficiently estimates the data density at each point – essential for local outlier detection. This is based on the observation that a cluster of points close to each other tend to have a similar influence on a target point's density estimation when used as kernel centers. These points thus can be represented by one abstract kernel center. Next, the KELOS's inlier pruning strategy early prunes points that have no chance to become top-N outliers. This empowers KELOS to skip the computation of their data density and of the outlier status for every data point. Together *a*KDE and the inlier pruning strategy eliminate the performance bottleneck of streaming local outlier detection. The experimental evaluation demonstrates that KELOS is up to 6 orders of magnitude faster than existing solutions, while being highly effective in detecting local outliers from streaming data.

1 INTRODUCTION

Motivation. The growth of digital devices coupled with their ever-increasing capabilities to generate and transmit live data presents an exciting new opportunity for real time data analytics. As the volume and velocity of data streams continue to grow, automated discovery of insights in such streaming data is critical to conform to the increasingly expected behavior exemplified by the new data. Thus, in streaming environments, it is critical to design a mechanism to efficiently identify outliers by monitoring the statistical properties of the data relative to each other as it changes over time.

State-of-the-Art. To satisfy this need, several methods [20, 21] have been proposed in recent years that leverage the concept of *local outlier* [6] to detect outliers from data streams. The local outlier notion is based on the observation that real world datasets tend to be skewed, where different subspaces of the data exhibit different distribution properties. It is thus often more meaningful to decide on the outlier status of a point based on its difference with the points in its local neighborhood as opposed to using a global density [9] or frequency [5] cutoff threshold to detect outliers [11]. More specifically, a point x is considered to be a *local outlier* if the *data density* at x is low *relative* to that at the points in x's local neighborhood. Unfortunately, existing streaming local outlier solutions [20, 21] are not scalable to high volume data streams. The root cause is that they measure the data density at each point x based on the point's distance to its k nearest neighbors (kNN). Unfortunately, kNN is very sensitive to data updates, meaning that the insertion or removal of even a small number of points can cause the kNN of many points in the dataset to be updated [20]. Since the complexity of the kNN search [6] is quadratic in the number of the points, significant resources may be wasted on a large number of unnecessary kNN re-computations. Therefore, those approaches suffer from a high response time when handling high-speed streams. For example, it takes [20, 21] 10 minutes to process just 100k tuples as shown by their experiments. Intuitively, kernel density estimation (KDE) [26], an established probability density approximation method, could be leveraged for estimating the data density at each point [16, 23, 27]. Unlike kNN-based density estimation that is sensitive to data changes, KDE estimates data density based on the statistical properties of the dataset. Therefore, it tends to be more robust to gradual data changes and thus

Problem

- Efficient outlier detection in stream data

Solution

- KELOS, a windowing-based algorithm that aggressively prunes non-outliers and calculates outliers based on their distance to kernels (clusters of high density)

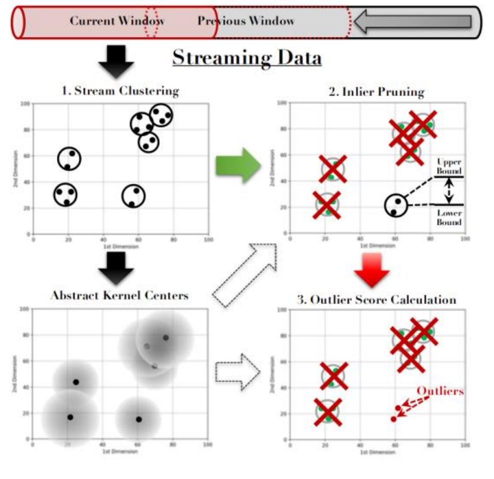
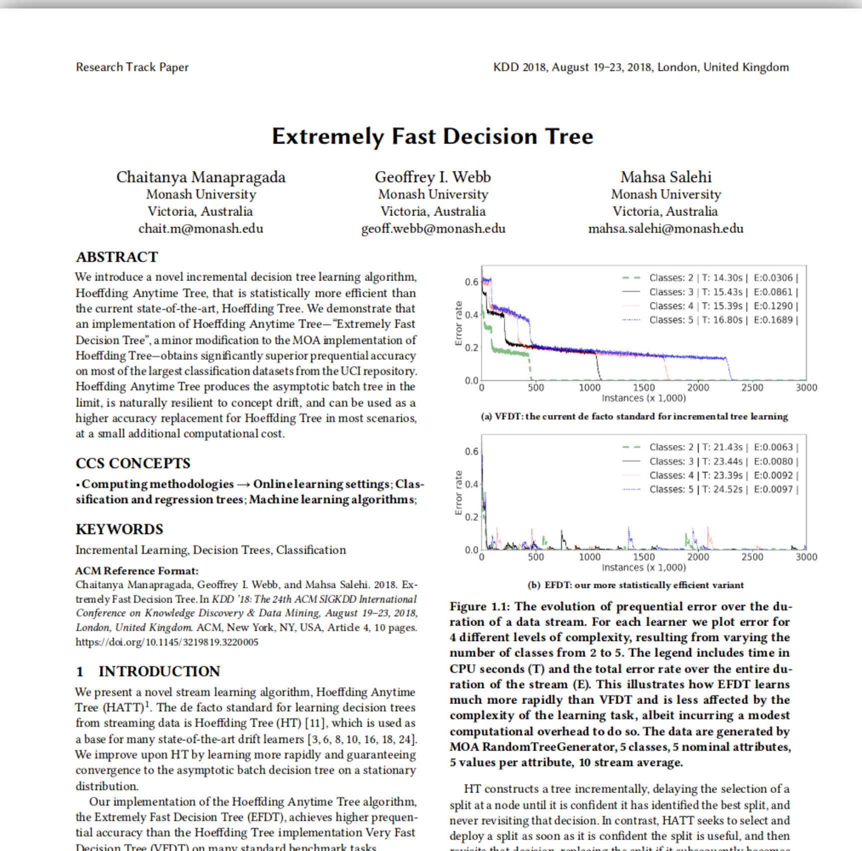


Figure 1: An illustration of KELOS approach.

Paper Suggestions



Research Track Paper KDD 2018, August 19-23, 2018, London, United Kingdom

Extremely Fast Decision Tree

Chaitanya Manapragada
Monash University
Victoria, Australia
chait.m@monash.edu

Geoffrey I. Webb
Monash University
Victoria, Australia
geoff.webb@monash.edu

Mahsa Salehi
Monash University
Victoria, Australia
mahsa.salehi@monash.edu

ABSTRACT

We introduce a novel incremental decision tree learning algorithm, Hoeffding Anytime Tree, that is statistically more efficient than the current state-of-the-art, Hoeffding Tree. We demonstrate that an implementation of Hoeffding Anytime Tree—"Extremely Fast Decision Tree", a minor modification to the MOA implementation of Hoeffding Tree—obtains significantly superior prequential accuracy on most of the largest classification datasets from the UCI repository. Hoeffding Anytime Tree produces the asymptotic batch tree in the limit, is naturally resilient to concept drift, and can be used as a higher accuracy replacement for Hoeffding Tree in most scenarios, at a small additional computational cost.

CCS CONCEPTS

• Computing methodologies → Online learning settings; Classification and regression trees; Machine learning algorithms;

KEYWORDS

Incremental Learning, Decision Trees, Classification

ACM Reference Format:
Chaitanya Manapragada, Geoffrey I. Webb, and Mahsa Salehi. 2018. Extremely Fast Decision Tree. In KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom. ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3219819.3220005

(a) VFDT: the current de facto standard for incremental tree learning

(b) EFDT: our more statistically efficient variant

Figure 1.1: The evolution of prequential error over the duration of a data stream. For each learner we plot error for 4 different levels of complexity, resulting from varying the number of classes from 2 to 5. The legend includes time in CPU seconds (T) and the total error rate over the entire duration of the stream (E). This illustrates how EFDT learns much more rapidly than VFDT and is less affected by the complexity of the learning task, albeit incurring a modest computational overhead to do so. The data are generated by MOA RandomTreeGenerator, 5 classes, 5 nominal attributes, 5 values per attribute, 10 stream average.

1 INTRODUCTION

We present a novel stream learning algorithm, Hoeffding Anytime Tree (HATT)[1]. The de facto standard for learning decision trees from streaming data is Hoeffding Tree (HT) [11], which is used as a base for many state-of-the-art drift learners [3, 6, 8, 10, 16, 18, 24]. We improve upon HT by learning more rapidly and guaranteeing convergence to the asymptotic batch decision tree on a stationary distribution.

Our implementation of the Hoeffding Anytime Tree algorithm, the Extremely Fast Decision Tree (EFDT), achieves higher prequential accuracy than the Hoeffding Tree implementation Very Fast Decision Tree (VFDT) on many standard benchmark tasks.

HT constructs a tree incrementally, delaying the selection of a split at a node until it is confident it has identified the best split, and never revisiting that decision. In contrast, HATT seeks to select and deploy a split as soon as it is confident the split is useful, and then revisits that decision, replacing the split if it subsequently becomes

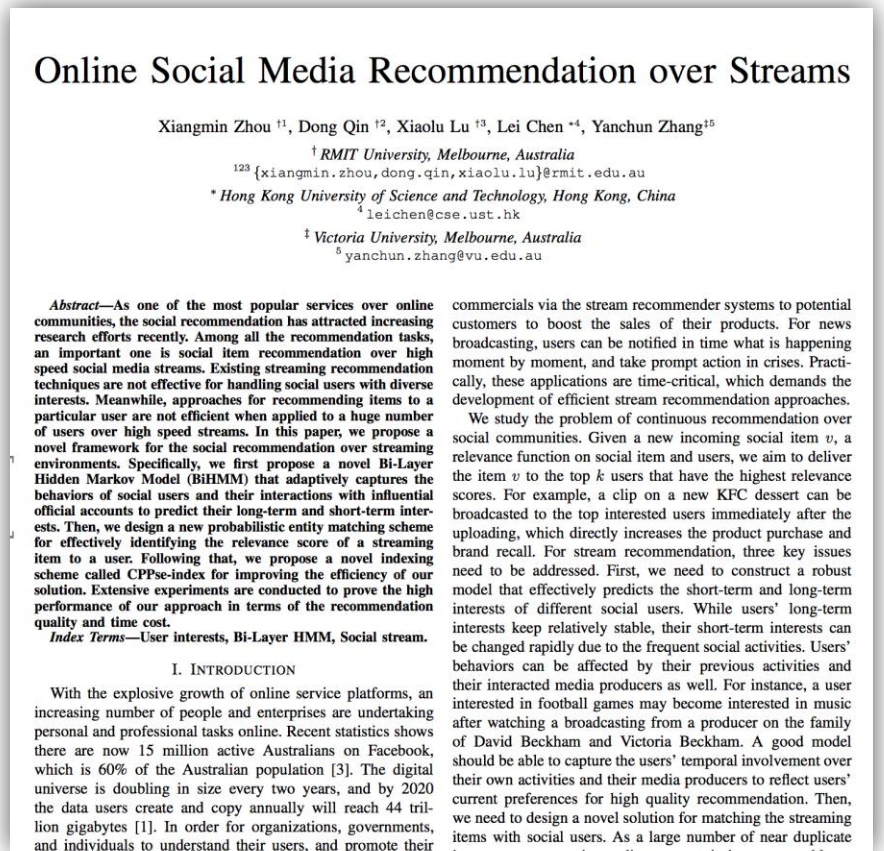
Problem

- Efficiently training a decision tree with streaming data

Solution

- Constructs a decision tree incrementally, based on the standard method for learning decision trees from streaming data, i.e., Hoeffding tree.
- Hoeffding trees exploit the fact that a small sample can often be enough to choose an optimal splitting attribute. This idea is supported mathematically by the Hoeffding bound, which quantifies the number of observations (in our case, examples) needed to estimate the goodness of a splitting attribute.
- The method in this papers achives higher accuracy for because splitting attributes will be replaced as soon as a better alternative is identified.

Paper Suggestions



Online Social Media Recommendation over Streams

Xiangmin Zhou †1, Dong Qin †2, Xiaolu Lu †3, Lei Chen *4, Yanchun Zhang‡5
† RMIT University, Melbourne, Australia
123 {xiangmin.zhou,dong.qin,xiaolu.lu}@rmit.edu.au
* Hong Kong University of Science and Technology, Hong Kong, China
4 leichen@cse.ust.hk
‡ Victoria University, Melbourne, Australia
5 yanchun.zhang@vu.edu.au

Abstract—As one of the most popular services over online communities, the social recommendation has attracted increasing research efforts recently. Among all the recommendation tasks, an important one is social item recommendation over high speed social media streams. Existing streaming recommendation techniques are not effective for handling social users with diverse interests. Meanwhile, approaches for recommending items to a particular user are not efficient when applied to a huge number of users over high speed streams. In this paper, we propose a novel framework for the social recommendation over streaming environments. Specifically, we first propose a novel Bi-Layer Hidden Markov Model (BiHMM) that adaptively captures the behaviors of social users and their interactions with influential official accounts to predict their long-term and short-term interests. Then, we design a new probabilistic entity matching scheme for effectively identifying the relevance score of a streaming item to a user. Following that, we propose a novel indexing scheme called CPPse-index for improving the efficiency of our solution. Extensive experiments are conducted to prove the high performance of our approach in terms of the recommendation quality and time cost.

Index Terms—User interests, Bi-Layer HMM, Social stream.

I. Introduction

With the explosive growth of online service platforms, an increasing number of people and enterprises are undertaking personal and professional tasks online. Recent statistics shows there are now 15 million active Australians on Facebook, which is 60% of the Australian population [3]. The digital universe is doubling in size every two years, and by 2020 the data users create and copy annually will reach 44 trillion gigabytes [1]. In order for organizations, governments, and individuals to understand their users, and promote their commercials via the stream recommender systems to potential customers to boost the sales of their products. For news broadcasting, users can be notified in time what is happening moment by moment, and take prompt action in crises. Practically, these applications are time-critical, which demands the development of efficient stream recommendation approaches.

We study the problem of continuous recommendation over social communities. Given a new incoming social item v, a relevance function on social item and users, we aim to deliver the item v to the top k users that have the highest relevance scores. For example, a clip on a new KFC dessert can be broadcasted to the top interested users immediately after the uploading, which directly increases the product purchase and brand recall. For stream recommendation, three key issues need to be addressed. First, we need to construct a robust model that effectively predicts the short-term and long-term interests of different social users. While users' long-term interests keep relatively stable, their short-term interests can be changed rapidly due to the frequent social activities. Users' behaviors can be affected by their previous activities and their interacted media producers as well. For instance, a user interested in football games may become interested in music after watching a broadcasting from a producer on the family of David Beckham and Victoria Beckham. A good model should be able to capture the users' temporal involvement over their own activities and their media producers to reflect users' current preferences for high quality recommendation. Then, we need to design a novel solution for matching the streaming items with social users. As a large number of near duplicate

Problem

- Social item (Youtube videos, news, tweets etc.) recommendation over high speed social media streams: Given a new incoming social item *v*, a relevance function on social item and users, we aim to deliver the item *v* to the top-k users that have the highest relevance scores.

Solution

- Novel Bi-Layer Hidden Markov Model (BiHMM) that adaptively captures the behaviors of social users and their interactions for predicting the users' long-term interest patterns
- A new probabilistic entity matching scheme for effectively identifying the relevance score of a streaming item to a user