Data Management
12 Stream Processing

Matthias Boehm
Graz University of Technology, Austria
Computer Science and Biomedical Engineering
Institute of Interactive Systems and Data Science
BMK endowed chair for Data Management

Last update: Jun 11, 2021
Announcements/Org

- **#1 Video Recording**
  - Link in TeachCenter & TUbe (lectures will be public)
  - [https://tugraz.webex.com/meet/m.boehm](https://tugraz.webex.com/meet/m.boehm)
  - Corona traffic light RED → May 17: ORANGE → Jul 01: YELLOW

- **#2 Reminder Communication**
  - **Newsgroup:** news://news.tugraz.at/tu-graz.lv.dbase
  - **Office hours:** Mo 12.30-1.30pm ([https://tugraz.webex.com/meet/m.boehm](https://tugraz.webex.com/meet/m.boehm))

- **#3 Exercises/Exams**
  - **Grading:** Exercise 1 – done, Exercise 2 – done
  - **Submission:** Exercise 3: start grading, Exercise 4: due Jun 22
  - **Exams:** Jun 30 5.30pm (i11, i12, i13), Jul 5 3.30pm (i13), 6.30pm (i13)

- **#4 Course Evaluation**
  - Please participate; open period: **June 1 – July 15**
Data Management Courses

- **Architecture of Database Systems** (ADBS, WS)
- **DB system internals** + prog. project

- **Architecture of ML Systems** (AMLS, SS)
- ML system internals
  - Prog. projects in [SystemDS](https://github.com/apache/systemds)

- **Data Integration and Large-Scale Analysis** (DIA, WS)
  - Distributed Data Management

- **Data Management / Databases** (DM, SS+WS)
  - Data management from user/application perspective

- **Intro to Scientific Writing** (WS)
Agenda

- Data Stream Processing
- Distributed Stream Processing
- Q&A and Exam Preparation

Data Integration and Large-Scale Analysis (DIA) (bachelor/master)
Data Stream Processing
Stream Processing Terminology

- **Ubiquitous Data Streams**
  - Event and message streams (e.g., click stream, twitter, etc)
  - Sensor networks, IoT, and monitoring (traffic, env, networks)

- **Stream Processing Architecture**
  - Infinite input streams, often with window semantics
  - Continuous (aka standing) queries
Stream Processing Terminology, cont.

- **Use Cases**
  - Monitoring and alerting (notifications on events / patterns)
  - Real-time reporting (aggregate statistics for dashboards)
  - Real-time ETL and event-driven data updates
  - Real-time decision making (fraud detection)
  - Data stream mining (summary statistics w/ limited memory)

- **Data Stream**
  - Unbounded stream of data tuples \( S = (s_1, s_2, \ldots) \) with \( s_i = (t_i, d_i) \)
  - See [10 NoSQL Systems](#) (time series)

- **Real-time Latency Requirements**
  - Real-time: guaranteed task completion by a given deadline (30 fps)
  - Near Real-time: few milliseconds to seconds
  - In practice, used with much weaker meaning
History of Stream Processing Systems

- **2000s**
  - **Data stream management systems** (DSMS, mostly academic prototypes):
    - STREAM (Stanford’01), Aurora (Brown/MIT/Brandeis’02) → Borealis (’05),
    - NiagaraCQ (Wisconsin), TelegraphCQ (Berkeley’03), and many others
    - but mostly unsuccessful in industry/practice
  - **Message-oriented middleware** and **Enterprise Application Integration** (EAI):
    - IBM Message Broker, SAP eXchange Infra., MS Biztalk Server, TransConnect

- **2010s**
  - **Distributed stream processing engines**, and “unified” batch/stream processing
  - **Proprietary systems**: Google Cloud Dataflow, MS StreamInsight / Azure Stream Analytics, IBM InfoSphere Streams / Streaming Analytics, AWS Kinesis
  - **Open-source systems**: Apache Spark Streaming (Databricks), Apache Flink (Data Artisans/Alibaba), Apache Beam, Apache Kafka, Apache Storm
# System Architecture – Native Streaming

## Basic System Architecture
- **Data flow graphs (potentially w/ multiple consumers)**
- **Nodes**: asynchronous ops (w/ state) (e.g., separate threads)
- **Edges**: data dependencies (tuple/message streams)
- **Push model**: data production controlled by source

## Operator Model
- Read from input queue
- Write to potentially many output queues
- Example Selection
  \[ \sigma_{A=7} \]

```java
while( !stopped ) {
    r = in.dequeue(); // blocking
    if( pred(r.A) ) // A==7
        for( Queue o : out )
            o.enqueue(r); // blocking
}
```
System Architecture – Sharing

- **Multi-Query Optimization**
  - Given *set of continuous queries* (deployed), compile DAG w/o redundancy (see 08 Physical Design MV) ➞ common subexpression elimination

- **Operator and Queue Sharing**
  - **Operator sharing**: complex ops w/ multiple predicates for adaptive reordering
  - **Queue sharing**: avoid duplicates in output queues via masks
System Architecture – Handling Overload

- **#1 Back Pressure**
  - Graceful handling of overload w/o data loss
  - **Slow down sources**
  - E.g., blocking queues

- **#2 Load Shedding**
  - **#1 Random-sampling**-based load shedding
  - **#2 Relevance-based** load shedding
  - **#3 Summary-based** load shedding (synopses)
  - Given SLA, select queries and shedding placement that minimize error and satisfy constraints

- **#3 Distributed Stream Processing** (see course DIA)
  - Data flow partitioning (distribute the query)
  - Key range partitioning (distribute the data stream)
Time (Event, System, Processing)

- **Event Time**
  - Real time when the event/data item was created

- **Ingestion Time**
  - System time when the data item was received

- **Processing Time**
  - System time when the data item is processed

- **In Practice**
  - Delayed and unordered data items
  - Use of heuristics (e.g., water marks = delay threshold)
  - Use of more complex triggers (speculative and late results)
Durability and Consistency Guarantees

- **#1 At Most Once**
  - “Send and forget”, ensure data is never counted twice
  - Might cause data loss on failures

- **#2 At Least Once**
  - “Store and forward” or acknowledgements from receiver, replay stream from a checkpoint on failures
  - Might create incorrect state (processed multiple times)

- **#3 Exactly Once**
  - “Store and forward” w/ guarantees regarding state updates and sent msgs
  - Often via dedicated transaction mechanisms
Window Semantics

- **Windowing Approach**
  - Many operations like joins/aggregation *undefined over unbounded streams*
  - Compute operations over *windows of time or elements*

- **#1 Tumbling Window**
  - Every data item is only part of a single window
  - Aka Jumping window

- **#2 Sliding Window**
  - Time- or tuple-based sliding windows
  - Insert new and expire old data items
Spark Streaming Example

// create spark context w/ batch interval 1s
sc = new JavaStreamingContext(conf, Durations.seconds(1));

// create DStream listening on socket (ip, port)
lines = sc.socketTextStream("localhost", 9999);

// traditional word count example on Dstream batches
JavaPairDStream<String, Integer> wordCounts = lines
    .flatMap(x -> Arrays.asList(x.split(" ")).iterator())
    .mapToPair(s -> new Tuple2<>(s, 1))
    .reduceByKey((i1, i2) -> i1 + i2);
wordCounts.print();

// extended word count example on Dstream windows
JavaPairDStream<String, Integer> wordCounts2 = lines
    .flatMap(x -> Arrays.asList(x.split(" ")).iterator())
    .mapToPair(s -> new Tuple2<>(s, 1))
    .reduceByKeyAndWindow((i1, i2) -> i1 + i2,
                           Durations.seconds(30), Durations.seconds(10));
Stream Joins

- **Basic Stream Join**
  - **Tumbling window:** use classic join methods
  - **Sliding window** (symmetric for both R and S)
    - Applies to arbitrary join pred
    - See 08 Query Processing (NLJ)

- **Excursus: How Soccer Players Would do Stream Joins**
  - **Handshake-join** w/ 2-phase forwarding

For each new r in R:
1. **Scan** window of stream S to find match tuples
2. **Insert** new r into window of stream R
3. **Invalidate** expired tuples in window of stream R

[Jens Teubner, René Müller: How soccer players would do stream joins. SIGMOD 2011]
Stream Joins, cont.

- **Double-Pipelined Hash Join**
  - Join of bounded streams (or unbounded w/ invalidation)
  - **Equi join predicate**, symmetric and non-blocking
  - For every incoming tuple (e.g. left): probe (right)+emit, and build (left)

\[
\begin{align*}
HR,\text{RID} & : 1, 1, 2, 7 \\
HS,\text{SID} & : 1, 7, 7 \\
\end{align*}
\]

[Zachary G. Ives, Daniela Florescu, Marc Friedman, Alon Y. Levy, Daniel S. Weld: An Adaptive Query Execution System for Data Integration. SIGMOD 1999]
Excursus: Example Twitter Heron

- **Motivation**
  - Heavy use of Apache Storm at Twitter
  - Issues: debugging, performance, shared cluster resources, back pressure mechanism

- **Twitter Heron**
  - API-compatible distributed streaming engine
  - De-facto streaming engine at Twitter since 2014

- **Dhalion (Heron Extension)**
  - Automatically reconfigure Heron topologies to meet throughput SLO

- Now back pressure implemented in Apache Storm 2.0 (May 2019)
Q&A and Exam Preparation

Basic focus: fundamental concepts and ability to apply learned techniques to given problems
Exam Logistics

- **Timing/Logistics**
  - Exam starts 10min after official start
  - 90min working time (plenty of time to think about answers)
  - **Write into the worksheet if possible**, additional paper allowed
  - Leave early if ready (usually starting after 45min)
  - Grading for Jun 30 / Jul 5 will happen early July

- **Covered Content**
  - **Must-have**: Data modeling/normalization, SQL query processing
  - Relational algebra, physical design, query and transaction processing
  - **DM only**: NoSQL, distributed storage and computation, streaming
Exam Logistics, cont.

- **Past Exams**
  - [https://mboehm7.github.io/teaching/ss20_dbs/index.htm](https://mboehm7.github.io/teaching/ss20_dbs/index.htm) (3+3)
  - [https://mboehm7.github.io/teaching/ss19_dbs/index.htm](https://mboehm7.github.io/teaching/ss19_dbs/index.htm) (3+3)
#1 Data Modeling

- **Task 1a: Specify the cardinalities in Modified Chen notation (8 Points)**
  - A hospital employs at least 4 nurses and has at least 8 patient rooms.
  - A nurse works in exactly one hospital and treats up to 16 patients.
  - A patient is treated by at least one but potentially many nurses.
  - Every patient has a room, a room belongs to exactly one hospital, and rooms are never shared by multiple patients.

![Database Diagram]

- **Hospital**
  - Has **Room** M
  - Employed by **Nurse** M

- **Room**
  - Located in **Patient** C
  - Has **TV**, **Size**

- **Nurse**
  - Treats **Patient** M
  - Has **Name**, **Degree**

- **Patient**
  - Has **Name**, **Phone**
  - Has **PID**

- **Country**, **City**, **HID**, **Phone**

- **1** represents 1:1, **M** represents M:1, **C** represents 1:M, **MC** represents M:M relationships.
#1 Data Modeling, cont.

- **Task 1b: Specify the cardinalities in \((\text{min}, \text{max})\) notation** (4 Points)
  - A hospital employs at least 4 nurses and has at least 8 patient rooms.
  - A nurse works in exactly one hospital and treats up to 16 patients.
  - A patient is treated by at least one but potentially many nurses.
  - Every patient has a room, a room belongs to exactly one hospital, and rooms are never shared by multiple patients.

Only provide answers you’re asked for!
#1 Data Modeling, cont.

- **Task 1c: Map the given ER diagram into a relational schema (10 points)**
  - Including data types, primary keys, and foreign keys

- **Solution**
  - `Hospitals`
    - `HID:int, phone:char(16), Country:varchar(64), City:varchar(64)`
  - `Nurses`
    - `NID:int, Name:varchar(64), Degree:varchar(32), HIDFK:int`
  - `Patient`
    - `PID:int, Name:varchar(64), Phone:char(16), RIDFK:int`
  - `Room`
    - `RID:int, TV:boolean, Size:int, HIDFK:int`
  - `Treated`
    - `NIDFK:int, PIDFK:int`
#1 Data Modeling, cont.

- **Task 1d**: Bring your schema in 3\(^{rd}\) normal form and explain why it is in 3NF *(12 points)*
  - Let Hospital.Phone and Patient.Phone be multi-valued attributes
  - Assume the functional dependency City → Country

- **Solution**
  - **Phones** (Number:char(16), HID\(^{FK}\):int, PID\(^{FK}\):int)
  - **Cities** (City:varchar(64), Country:varchar(64))
  - **Hospitals** (HID:int, City\(^{FK}\):varchar(64))

- **1\(^{st}\) Normal Form**: no multi-valued attributes
- **2\(^{nd}\) Normal Form**: 1NF + all non-key attributes fully functional dependent on PK
- **3\(^{rd}\) Normal Form**: 2NF + no dependencies among non-key attributes
#2 Structured Query Language

- Task 2a: Compute the results for the following queries (15 points)

**Q1:** SELECT DISTINCT Customer, Date 
FROM Orders O, Products P 
WHERE O.PID = P.PID AND Name IN('Y','Z')

**Q2:** SELECT Customer, count(*) FROM Orders 
GROUP BY Customer 
ORDER BY count(*) DESC, Customer ASC

**Q3:** SELECT Customer, sum(O.Quantity * P.Price) 
FROM Orders O, Products P 
WHERE O.PID = P.PID 
GROUP BY Customer
#2 Structured Query Language, cont.

- **Task 2b: Write SQL queries to answer the following Qs (15 points)**

**Q4**: Which products where bought on 2019-06-25 (return the distinct product names)?

**Q5**: Which customers placed only one order?

**Q6**: How much revenue (sum( O.Quantity * P.Price)) did products with a price less than 90 generate (return (product name, revenue))?
#3 Query Processing

- **Task 3a:** Assume tables \( R(a,b) \), and \( S(c,d,e) \), draw a logical query tree in relational algebra for the following query: (5 points)

  \[
  Q7: \text{SELECT } R.a, S.d \text{ FROM } R, S \text{ WHERE } R.b = S.c \text{ AND } S.e < 3 \\
  \text{UNION ALL} \\
  \text{SELECT } R.a, S.d \text{ FROM } R, S \text{ WHERE } R.b = S.c \text{ AND } S.e = 7
  \]

- **Task 3b:** Draw an optimized logical query tree for the above query in relational algebra by eliminating the union operation (3 points)
#3 Query Processing, cont.

- **Task 3c**: Given the schema and query above, which attribute or attributes are good candidates for secondary indexes and how could they be exploited during query processing? *(4 points)*

- **Solution**
  - **S.e** → index scan
    - (lookup e=7, lookup e=3 and scan DESC)
  - **R.b (or S.c)** → index nested loop join
    - (for every S tuple s, lookup s.c in IX)

```
\[ \pi_{a,d} (R \bowtie_{b=c} S) \]
```
#3 Query Processing, cont.

- Task 3d: Describe the volcano (open-next-close) iterator model by example of a selection operator and discuss the space complexity of this selection operator. *(6 points)*

- Solution
  - Open, next, close calls propagate from root to leaves
  - **Open**: operator initialization
  - **Next**: compute next tuple (selection: call next of input until next qualifying tuple found)
  - **Close**: cleanup resources
  - **Space complexity**: $O(1)$

```c
void open() { R.open(); }
void close() { R.close(); }
Record next() {
    while( (r = R.next()) != EOF )
        if( p(r) ) //A==7
            return r;
    return EOF;
}
```
#4 Physical Design – B-Trees

- **Task 4a:** Given B-tree, *insert key 19* and draw resulting B-tree (7 points)

  \[
  k = 2 \\
  \begin{array}{c}
  2 \ 5 \ 8 \\
  11 \ 17 \\
  21 \ 23 \ 25 \ 29 \\
  \end{array} \\
  \begin{array}{c}
  9 \ 18 \\
  \end{array} \\
  \begin{array}{c}
  2 \ 5 \ 8 \\
  11 \ 17 \\
  21 \ 23 \ 25 \ 29 \\
  9 \ 18 \ 23 \\
  \end{array}
  \]

- **Task 4b:** Given B-tree, *delete key 27*, and draw resulting B-tree (8 points)

  \[
  k = 2 \\
  \begin{array}{c}
  2 \ 5 \\
  11 \ 16 \ 19 \\
  23 \ 25 \\
  29 \ 30 \\
  \end{array} \\
  \begin{array}{c}
  8 \ 20 \ 27 \\
  \end{array} \\
  \begin{array}{c}
  8 \ 20 \\
  2 \ 5 \\
  11 \ 16 \ 19 \\
  23 \ 25 \ 29 \ 30 \\
  \end{array}
  \]
Task 5a: Describe the concept of a database transaction log, and explain how it relates to the ACID properties Atomicity and Durability (7 points)

Solution

- Log: append-only TX changes, often on separate devices
- Write-ahead logging (log written before DB, forced-log on commit)
- Recovery: forward (REDO) and backward (UNDO) processing

#1 Atomicity: A TX is executed atomically (completely or not at all); on failure/aborts no changes in DB (UNDO)

#2 Durability: Guaranteed persistence of changes of successful TXs; in case of system failures, the database is recoverable (REDO)
#6 NoSQL

- **Task 6a:** Describe the concept and system architecture of a **key-value store**, including techniques for achieving **high write throughput**, and **scale-out** in distributed environments. Please focus specifically on aspects of physical design such as **index structures**, and **distributed data storage**. (10 points)

- **Solution**
  - **KV store:** simple map of key-value pairs, w/ get/put interface, often distributed
  - **Index structure for high write throughput:** Log-structured merge trees (LSM)
  - **Distributed data storage for scale-out:** horizontal partitioning (sharding) via hash or range partitioning, partitioning via selection, reconstruction via union eventual consistency for high availability and partition tolerance
Conclusions and Q&A

- 12 Stream Processing Systems
- Q&A and Exam Preparation

Misc
- **Last office hours:** Jun 21, 12.30pm / Jun 28, 12.30pm
- Exercise 4 Reminder: **Jun 22, 11.59pm** + [7 late days]

Exams
- **Exams:** Jun 30 5.30pm (i11, i12, i13), Jul 5 3.30pm (i13), 6.30pm (i13)
- **Oral exams** for international students (so far 2, to be scheduled)