Ultra-Low Latency with Java and Terabytes of Data

Per Minborg
CTO, Speedment, Inc.
Ultra-Low latency = 200 ns
Why Are Applications Slow?

- Slow Databases
- Data on Several Nodes and no Affinity Across Data
- Data is Remote
- Unnecessary Object Creation / Garbage Collect Problem
- Lack of Parallelism
Why Are Applications Slow?
Slow Databases

Data grows exponentially, which clogs systems
Why Are Applications Slow?
Several Nodes/no Affinity Across Data
Why Are Applications Slow?
Several Nodes/no Affinity Across Data

Low Latency

Low Affinity

Scale Out

[Diagram showing the relationship between low latency, low affinity, and scale out]
Why Are Applications Slow?  Several Nodes/no Affinity Across Data

Σ=10  Σ=20  Σ=30
Why Are Applications Slow?
Several Nodes/no Affinity Across Data

Σ=10  Σ=20  Σ=30
Why Are Applications Slow?
Several Nodes/no Affinity Across Data
Why Are Applications Slow?
Several Nodes/no Affinity Across Data
Why Are Applications Slow?

Several Nodes / no Affinity Across Data
Why Are Applications Slow?
Several Nodes/no Affinity Across Data
Why Are Applications Slow?
Data is Remote: Laws of Nature

45 ms

100 us

25 us
Why Are Applications Slow?  
Data is Remote: Operating System

1-3 us

Process

Process

Process

Process

Linux Kernel
Why Are Applications Slow?
Unnecessary Object Creation

1 s
Why Are Applications Slow? 
Unnecessary Object Creation

To write a single Java object to main memory takes 200 ns

Conclusion: Creating shared objects -> not ultra-low latency
Why Are Applications Slow? 
Lack of Parallelism

```
$ nproc --all
32

$ top

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>%CPU</th>
<th>%MEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2105</td>
<td>java</td>
<td>100.0</td>
<td>5.4</td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>0.5</td>
<td>0.4</td>
</tr>
</tbody>
</table>
```
The Solution: In-JVM-Memory

What is That?
In-Memory vs. In-JVM-Memory

**In-Memory**
- Data is in RAM
- The application is remotely connected to a grid, other machine, other process

**In-JVM-Memory**
- Data is in RAM
- The application and data resides in the same JVM
In-JVM-Memory Makes Ultra Low Latency Possible

• CPU Cache Latencies:
  • L1 ~0.5 ns
  • L2 ~7 ns
  • L3 ~20 ns
• 64-bit Main Memory Read ~100 ns
In-JVM-Memory vs. In-Memory Performance
In-JVM-Memory Scalability

Is That Even Possible?
Scaling up In-JVM-Memory

Today: Scale up to 12 TB (Intel® Xeon® Processor E7-8855 v4 * 4)

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>Memory</th>
<th>Logical Processors</th>
<th>Dedicated EBS Bandwidth</th>
<th>Network Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>u-6tb1.metal</td>
<td>6 TiB</td>
<td>448</td>
<td>14 Gbps</td>
<td>25 Gbps</td>
</tr>
<tr>
<td>u-9tb1.metal</td>
<td>9 TiB</td>
<td>448</td>
<td>14 Gbps</td>
<td>25 Gbps</td>
</tr>
<tr>
<td>u-12tb1.metal</td>
<td>12 TiB</td>
<td>448</td>
<td>14 Gbps</td>
<td>25 Gbps</td>
</tr>
</tbody>
</table>

Soon: Scale up to 48 TB
• Increase Memory in the Cloud as You Grow
Is Scale Up Cost Effective?

**General Belief**

**Fact**

AWS "x1e.Nxlarge" Series: Memory Size vs. Cost

- Internal RAM [GiB]
- Cost [$/h]
What if I Have More Than 12 TB?

• High Level Sharding
  • Per year, region, segment

• Memory Mapping (e.g. IMDT)

• Use in-JVM-memory solution as an add on for your current solution for part of your data
What if My Data Grows?

- JVM RAM Size [GB]
- Your Data [GB]
In-JVM-Memory vs. In-Memory Performance

Data with 75% correlation

![Graph comparing In-JVM-Memory and Grid (25 GB/node) performance](image-url)
Recap: Impossible to Scale Out AND Get Low Latency When You Have Low Affinity
Solution: In-JVM Memory

- Scale Up
- Low Latency
- Low Affinity

In-JVM Memory Speedment
In-JVM-Memory Solution: Speedment
Speedment: In-JVM-Memory DataStore

- Continuously creates data snapshots from a data source
- Places the copy within the JVM
- Off-Heap Data
- Off-Heap Indexing
- No Impact on Garbage Collect
- Supports off-heap joins and aggregations
- Can operate without creating intermediary objects
java.util.stream.Stream
Database Actions Using Java 8 Stream Syntax Instead of SQL

Why should you need to use SQL when the same semantics can be derived directly from Java 8 streams? If you take a closer look at this objective, it turns out there is a remarkable resemblance between the verbs of Java 8 streams and SQL commands, as summarized in Table 1.

Streams and SQL queries have similar syntax in part because both are declarative constructs, meaning they describe a result rather than state instructions on how to compute the result. But as a SQL query describes a result set rather than the operations needed to compute the result, a Java stream describes the result of a sequence of abstract functions without dictating the properties of the actual computation.

The open-source project Speedment capitalizes on this similarity to enable you to perform database actions using Java 8 stream syntax instead of SQL. It is available on GitHub under the business-friendly Apache 2 license for open-source databases. A license fee is required for commercial databases. Find free to close the entire project.

About Speedment

Speedment allows you to write pure Java code for entire database applications. It uses lazy evaluation of streams, meaning that only a minimum set of data is actually pulled from the database into your application and only as the elements are needed.

<table>
<thead>
<tr>
<th>SQL COMMAND</th>
<th>JAVA 8 STREAM OPERATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM</td>
<td>streams()</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>map()</td>
</tr>
<tr>
<td>HAVING</td>
<td>filter(.identity())</td>
</tr>
<tr>
<td>JOIN</td>
<td>flatMap() OR map()</td>
</tr>
<tr>
<td>DISTINCT</td>
<td>distinct()</td>
</tr>
<tr>
<td>UNION</td>
<td>concat(s, s).distinct()</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>sort()</td>
</tr>
<tr>
<td>LIMIT</td>
<td>take()</td>
</tr>
<tr>
<td>DROP BY</td>
<td>collect(groupingBy())</td>
</tr>
<tr>
<td>COUNT</td>
<td>count()</td>
</tr>
</tbody>
</table>

Table 1. SQL commands and their counterpart verbs in Java 8 streams
Declarative Constructs in SQL and Stream

```
SELECT * FROM FILM
    WHERE RATING = 'PG-13'
```

```
films.stream()
    .filter(Film.RATING.equal(Rating.PG13))
```
Speedment Can Process Data without Creating Intermediate Objects

```java
films.stream()
    .filter(Film.RATING.equal(Rating.PG13))
    .count();
```
Speedment Can Process Data without Creating Intermediate Objects

```java
films.stream()
    .filter(Film.RATING.equal(Rating.PG13))
    .collect(toJsonLengthAndTitle());
```
var join = joinComponent
  .from(FilmManager.IDENTIFIER)
  .innerJoinOn(Language.LANGUAGE_ID).equal(Film.LANGUAGE_ID)
  .build(Tuples::of);
```java
var offHeapAggregator = Aggregator.builder(Result::new)
    .on(Language.LANGUAGE_ID).key(Result::setLanguage)
    .on(Film.RATING).key(Result::setRating)
    .on(Film.LENGTH).average(Result::setAverage)
    .build();
```
var result = join.stream()
  .collect(offHeapAggregator);
join.stream()
  .parallel()
  .collect(offHeapAggregator);
$ nproc -all
32

$ top

+-----------------+-----+-----+
| PID  | USER | %CPU | %MEM |
+-----------------+-----+-----+
| 2107 | java | 3170.0 | 5.4 |
| 1    | root | 0.5  | 0.4  |
+-----------------+-----+-----+
Hands on Demo

Seeing is Believing
@Benchmark
public long filterAndCount() {
    return films.stream()
        .filter(RATING_EQUALS_PG_13)
        .count();
}
Demo: Download Sakila Demo Database

MySQL Help Tables

<table>
<thead>
<tr>
<th>Title</th>
<th>Version</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL Help Tables</td>
<td>8.0</td>
<td>Gzip</td>
</tr>
<tr>
<td>MySQL Help Tables</td>
<td>5.7</td>
<td>Gzip</td>
</tr>
<tr>
<td>MySQL Help Tables</td>
<td>5.6</td>
<td>Gzip</td>
</tr>
<tr>
<td>MySQL Help Tables</td>
<td>5.5</td>
<td>Gzip</td>
</tr>
</tbody>
</table>
Demo: Initialize the Project
Demo: Connect to the Sakila Database
Demo: Generate the Domain Model
Use Existing Infrastructure

How does it Fit with What We Have?
Easy Integration: Any Data Source

DATABASE

ORACLE

IBM

DB2

IBM

AS/400

PostgreSQL

CSV

BIN

MariaDB

Google Sheets

AVRO

MySQL

SQL Server

CONNECT TO DATABASE

Database Type: MySQL

Database Host: 127.0.0.1

Username: root

Password: ********

Database Name: sakila
Deploy Anywhere

On Premise

IBM Bluemix
Amazon Web Services
Microsoft Azure
Oracle Cloud
Docker
Google Cloud Platform
Kubernetes
IDE Integration

- eclipse
- IntelliJ (IJ)
- NetBeans
Web Service Integration

- Spring
- Oracle Weblogic Server
- Payara
- Apache TomEE
- WildFly
- GlassFish
- JBoss by Red Hat
Thanks

Trial License? Contact:

Per Minborg
minborg@speedment.com

www.speedment.com/initializer

github.com/speedment/speedment
films.stream()
  .filter(Film.RATING.equal(Rating.PG13))
  .collect(toJsonLengthAndTitle()));

<table>
<thead>
<tr>
<th>index</th>
<th>film_id</th>
<th>length</th>
<th>rating</th>
<th>year</th>
<th>language</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>0</td>
<td>267</td>
<td>267</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>[1]</td>
<td>267</td>
<td>0</td>
<td>0</td>
<td>267</td>
<td>267</td>
<td>267</td>
</tr>
<tr>
<td>[2]</td>
<td>523</td>
<td>523</td>
<td>523</td>
<td>523</td>
<td>523</td>
<td>523</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>index</th>
<th>film_id</th>
<th>length</th>
<th>rating</th>
<th>year</th>
<th>language</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>1</td>
<td>123</td>
<td>PG-13</td>
<td>2006</td>
<td>1</td>
<td>ACAD..</td>
</tr>
<tr>
<td>[267]</td>
<td>2</td>
<td>69</td>
<td>G</td>
<td>2006</td>
<td>1</td>
<td>ACE G..</td>
</tr>
<tr>
<td>[523]</td>
<td>3</td>
<td>134</td>
<td>PG-13</td>
<td>2006</td>
<td>1</td>
<td>ADAP...</td>
</tr>
</tbody>
</table>