GraalVM at Facebook

Chen Li
2021 Graal Workshop

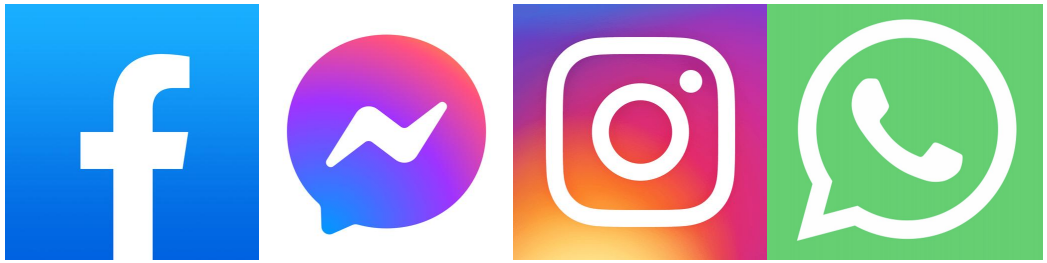
About me...

- Software Engineer at Facebook (Programming Languages & Runtimes Team)
- Focus on Java Efficiency
- Previously at LinkedIn and Azul Systems

Agenda

- Java at Facebook
- Why GraalVM
- Spark on GraalVM
- GraalVM Bugs
- Future Plan

Facebook



1.84 Billion DAU
2.80 Billion MAU

Source: Facebook Q4 2020 earnings call

Where is Java used?

- Big Data Services
  - Spark, Presto, etc.
- Backend Services
- Mobile: Android

Where is Java used?

- Big Data Services
  - Spark, Presto, etc.
- Backend Services
- Mobile: Android

Facebook's JDK

- Oracle Hotspot & OpenJDK for Java 8
- OpenJDK for Java 11
- No customization

Why Graal?

- Performance
  - Better optimizations i.e. escape analysis
  - YoY improvements
- Easier to learn Graal than C2
- Community

How we use Graal?

- GraalVM CE
- We use Graal as a JIT compiler to replace C2:

```
java -XX:+UnlockExperimentalVMOptions -XX:+EnableJVMCI -XX:+UseJVMCICompiler
```

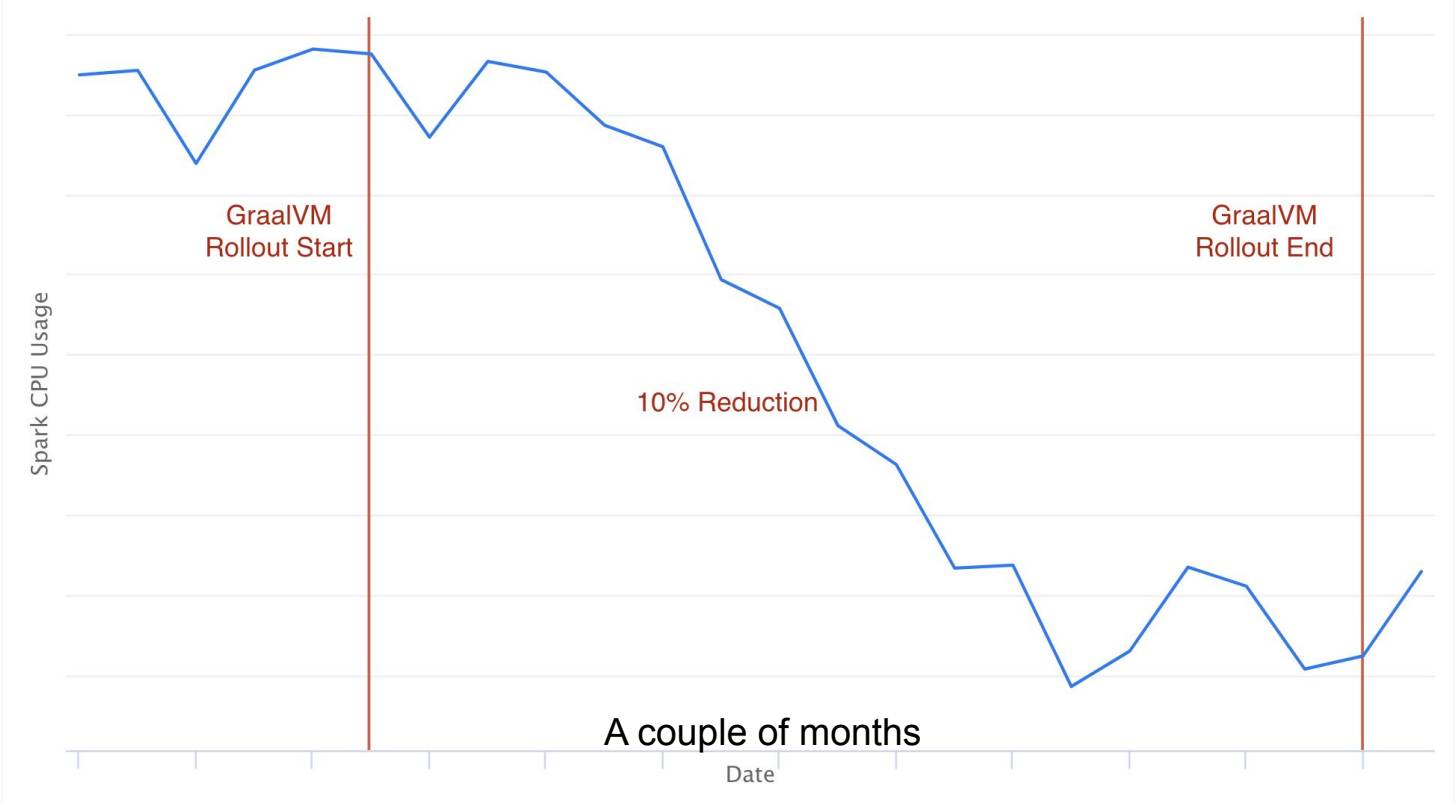
Spark at Facebook

- Largest SQL query engine at Facebook
- Run on disaggregated compute/storage clusters
- Efficiency is high priority

Spark on GraalVM

- Evaluated in early 2020
- Started with local benchmarks and small test suites
- Rolled out to production in a couple of months
- No reliability issues except for one compiler bug

Spark on GraalVM



Which optimization matters?

- Polymorphic inlining
- Escape analysis
- Speculative optimizations

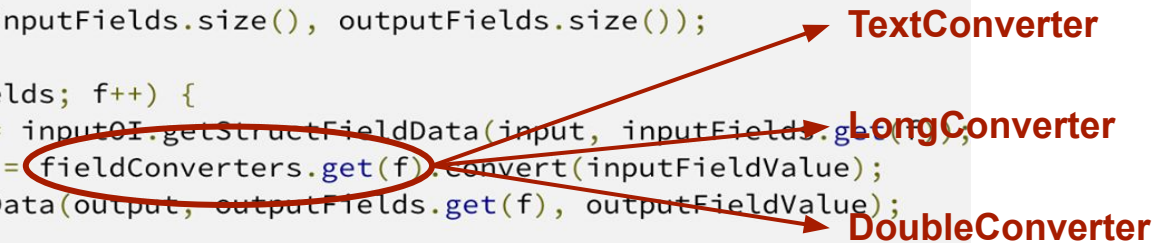
Polymorphic Inlining

```
public Object convert(Object input) {
  if (input == null) {
    return null;
  }

  int minFields = Math.min(inputFields.size(), outputFields.size());
  // Convert the fields
  for (int f = 0; f < minFields; f++) {
    Object inputFieldValue = inputOI.getStructFieldData(input, inputFields.get(f));
    Object outputFieldValue = fieldConverters.get(f).convert(inputFieldValue);
    outputOI.setStructFieldData(output, outputFields.get(f), outputFieldValue);
  }

  // set the extra fields to null
  for (int f = minFields; f < outputFields.size(); f++) {
    outputOI.setStructFieldData(output, outputFields.get(f), null);
  }

  return output;
}
```



Escape Analysis

- **Reduce object allocations**
- **Avoid boxing/unboxing**
- **5X less CPU consumption of** java/lang/Double.valueOf **in profiling results**

Speculative Optimizations

```
public Object get(int ordinal, DataType dataType) {
  if (isNullAt(ordinal) || dataType instanceof NullType) {
    return null;
  } else if (dataType instanceof BooleanType) {
    return getBoolean(ordinal);
  } else if (dataType instanceof ByteType) {
    return getByte(ordinal);
  } else if (dataType instanceof ShortType) {
  .... // many else if (dataType instanceof xxxType)
  } else if (dataType instanceof MapType) {
    return getMap(ordinal);
  } else if (dataType instanceof UserDefinedType) {
    return get(ordinal, ((UserDefinedType)dataType).sqlType());
```

Long if-else chains

Not as good

- JVMCI overhead
- Missing intrinsics
  i.e. Arrays.fill
- Inlining not always beats C2
- No auto vectorization in CE

What else did we try?

- Auto-tune compiler flags using Ax
- Spark code generation using Truffle framework

Auto-tune compiler flags using Ax

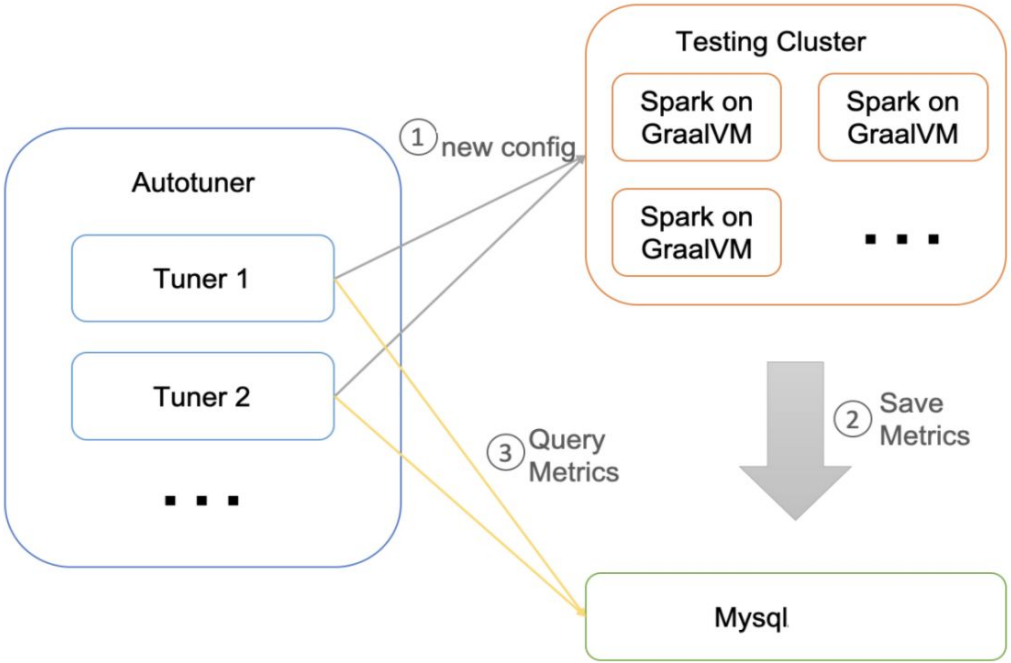
- **What is Ax?**

  Ax is a machine learning system that can optimize discrete configurations using multi-armed bandit optimization, and continuous configurations using Bayesian optimization.

  https://ax.dev/
- **What flags did we tune?**

  TrivialInliningSize, MaximumInliningSize, etc.

Auto-tune compiler flags using Ax



Auto-tune compiler flags using Ax

- Experiments show that it has potential to improve CPU performance for Spark workload, with over 10% on specific settings.
- Need to address some issues before moving to production
  - make tuning faster
  - reduce noise in performance measurement

Spark code generation using Truffle

- Use Truffle to write some SQL operators in Spark
- Early experiment: only a few operators were prototyped and no plan for production yet

Spark code generation using Truffle

We had some discussion with people doing similar projects

Dynamic Speculative Optimizations for SQL Compilation in Apache Spark

Filippo Schiavio
Università della Svizzera italiana (USI)
Switzerland
filippo.schiavio@usi.ch

Daniele Bonetta
VM Research Group
Oracle Labs
USA
daniele.bonetta@oracle.com

Walter Binder
Università della Svizzera italiana (USI)
Switzerland
walter.binder@usi.ch

ABSTRACT

Big-data systems have gained significant momentum, and Apache Spark is becoming a de-facto standard for modern data analytics. Spark relies on SQL query compilation to optimize the execution performance of analytical workloads on a variety of data sources. Despite its scalable architecture, Spark's SQL code generation suffers from significant runtime overheads related to data access and de-serialization. Such performance penalty can be significant, especially when applications operate on human-readable data formats such as CSV or JSON.

In this paper we present a new approach to query compilation that overcomes these limitations by relying on runtime profiling and dynamic code generation. Our new SQL compiler for Spark produces highly-efficient machine code, leading to speedups of up to 4.4x on the TPC-H benchmark with textual-form data formats such as CSV or JSON.

PVLDB Reference Format:
Filippo Schiavio, Daniele Bonetta, Walter Binder. Dynamic Speculative Optimizations for SQL Compilation in Apache Spark. *PVLDB*, 13(5): 754-767, 2020.
DOI: https://doi.org/10.14778/3377369.3377382

the context of large statistical analyses (expressed in Python or R). Furthermore, due to the growing popularity of data lakes [12], the interest in efficient solutions to analyze text-based data formats such as CSV and JSON is increasing even further.

At its core, the SQL language support in Spark relies on a managed language runtime – the Java Virtual Machine (JVM) – and on query compilation through so-called *whole-stage* code generation [7]. Whole-stage code generation in Spark SQL is inspired by the data-centric produce-consume model introduced in Hyper [23], which pioneered pipelined SQL compilation for DBMSs. Compiling SQL to optimize runtime performance has become common in commercial DBMSes (e.g., Oracle RDBMS [28], Cloudera Impala [42], PrestoDB [40], MapDB [38], etc.). Unlike traditional DBMSes, however, Spark SQL compilation does not target a specific data format (e.g., the columnar memory layout used by a specific database system), but targets *all* encoding formats supported by the platform. In this way, the same compiled code can be re-used to target multiple data formats such as CSV or JSON, without having to extend the SQL compiler back-end for new data formats. Thanks to this approach,

Bugs

- #2493 GraalVM generates wrong result due to speculative optimization
- #2869 GraalVM: JVMCI-native CompilerThreads are RUNNABLE but not get processed

Bug #2493

- https://github.com/oracle/graal/issues/2493
- Graal does not handle signed comparison and unsigned comparison correctly when checking for disjoint conditions, which results in wrong If-statement reordering.

Bug #2493

```
if (x < 0) {
    // must be empty so that it will merge with 'else' branch of
    // '(x < positive_constant)' w/o doing anything
} else {
    if (x < positive_constant) {
        never_executed_path;
    } else {
    }
}
if (x < positive_constant) {
    execute_true_path;
} else {
    execute_false_path;
}
```

Bug #2493

```
if (x < 0) {
    // must be empty so that it will merge with 'else' branch of
    // '(x < positive_constant)' w/o doing anything
} else {
    if (x < positive_constant) {
        uncommon_trap;
    } else {
    }
}
if (x < positive_constant) {
    execute_true_path;
} else {
    execute_false_path;
}
```

Bug #2493

```
if (unsigned_x < positive_constant) {
    uncommon_trap;
} else {
    if (x < positive_constant) {
        execute_true_path;
    } else {
        execute_false_path;
    }
}
```

Bug #2493

```
if (x < positive_constant) {
    execute_true_path;
} else {
    // bug: these two conditions should not be reordered.
    // After reordering, this condition would never happen
    if (unsigned_x < positive_constant) {
        uncommon_trap;
    } else {
        execute_false_path;
    }
}
```

Bug #2493

Closed **GraalVM generates wrong result due to speculative optimization** #2493

helloguo opened this issue on May 22, 2020 · 16 comments

lic9 commented on May 24, 2020 • edited

@dougxc @tkrodriguez canonicalizeConditionalCascade also uses "join". But as commented, it intentionally makes unsigned and signed return null for "join", so the logic is correct in **canonicalizeConditionalCascade**.

I couldn't find any other usage of "join".

tkrodriguez commented on May 29, 2020 Member

Yes the other cases are explicitly safe since they treat the null as meaning unknown. Fixed in `2dfa6ab`

👍 1

Bug #2869

- https://github.com/oracle/graal/issues/2869
- JVMCI-native CompilerThreads halted and no new methods were compiled
- Can not reproduce locally
- Workaround: disable libgraal

Bug #2869

Open **GraalVM: JVMCI-native CompilerThreads are RUNNABLE but not get processed** #2869

lic9 opened this issue on Sep 23, 2020 · 6 comments

We were running some Presto jobs on GraalVM and we found on 1 of the worker node, Graal stopped compiling new methods for more than 1 day. From thread dump, we saw all JVMCI-native CompilerThreads were at RUNNABLE but not consume any CPU ('cpu' metric from thread dump stayed the same and only 'elapsed' increased). The issue seemed only apply to JVMCI-native CompilerThreads. C1 CompilerThread was normal and tiered 3 C1 compilations were triggered for those new methods. The worker process had a few thousands of threads, and JVMCI-native CompilerThreads and C1 CompilerThread seem have the same thread priority.

We don't have a way to reproduce the issue locally.

- GraalVM version: GraalVM CE 20.2.0
- JDK Major Version: 11
- OS: CentOS
- Architecture: AMD64

Now and in the future

- Most CPU-bound big data services on GraalVM
  - Presto has > 5% CPU improvement and GC pause reduction switching to GraalVM
- Pushing for memory-bound services
  - Shenandoah GC, ZGC support
- Leveraging other features like native image
- Adding customized optimization passes for Facebook workloads

Community contribution

- Open to contribute patches to open source
- Open to ideas/collaborations to make community better

Thank you!