Distributed Refactoring across GitHub

Jon Schneider
Spring Team @ Pivotal, Inc.
@jon_k_schneider

```
1  package io.jschneider.gradlesummit.rewrite;
2
3  import com.google.common.base.Objects;
4
5  public class A {
6      void foo() {
7          Objects.firstNonNull(null, 1);
8      }
9  }
```

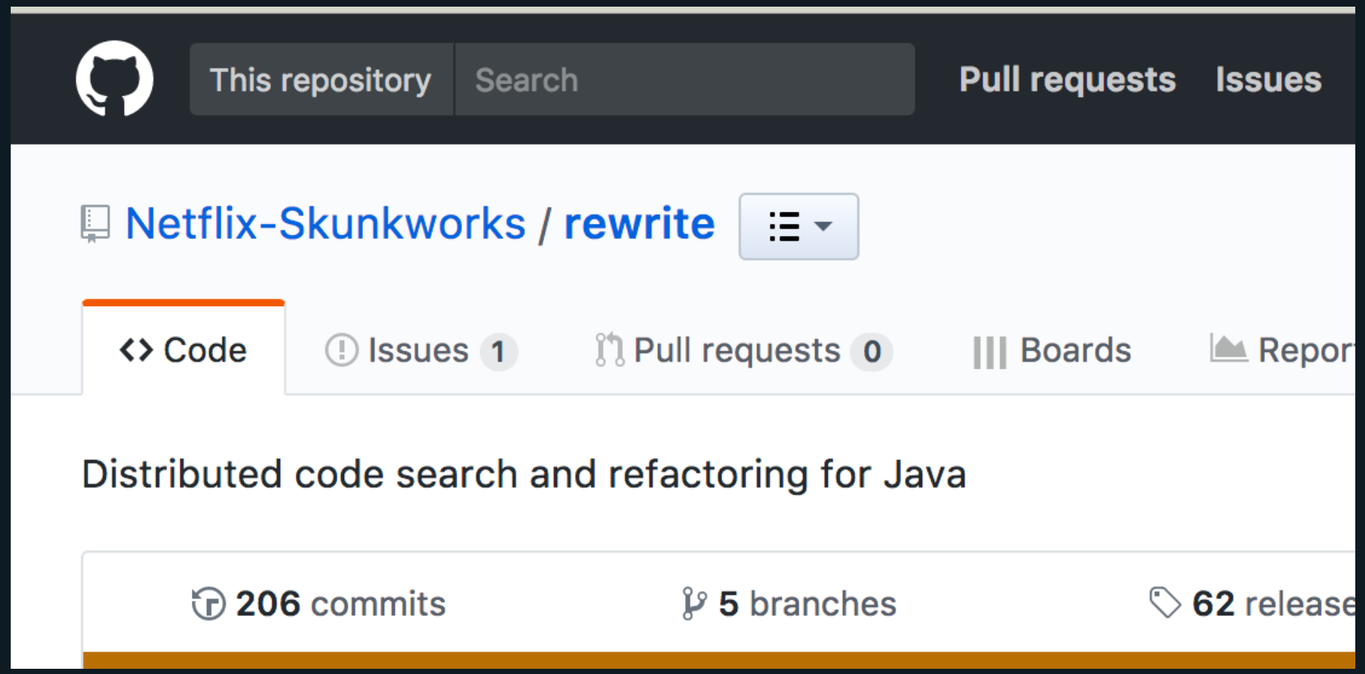
github.com/jkschneider/gradle-summit-2017

We will tackle this in 3 parts.

1. Netflix Rewrite to refactor them
2. Google BigQuery to find Java files
3. Zeppelin/Spark on Dataproc to run at scale

Part 1: Rewriting code

Rewrite is a programmatic refactoring tool.



Suppose we have a simple class A.

```
1  package io.jschneider.gradlesummit.rewrite;
2
3  import com.google.common.base.Objects;
4
5  public class A {
6      void foo() {
7          Objects.firstNonNull(null, 1);
8      }
9  }
```

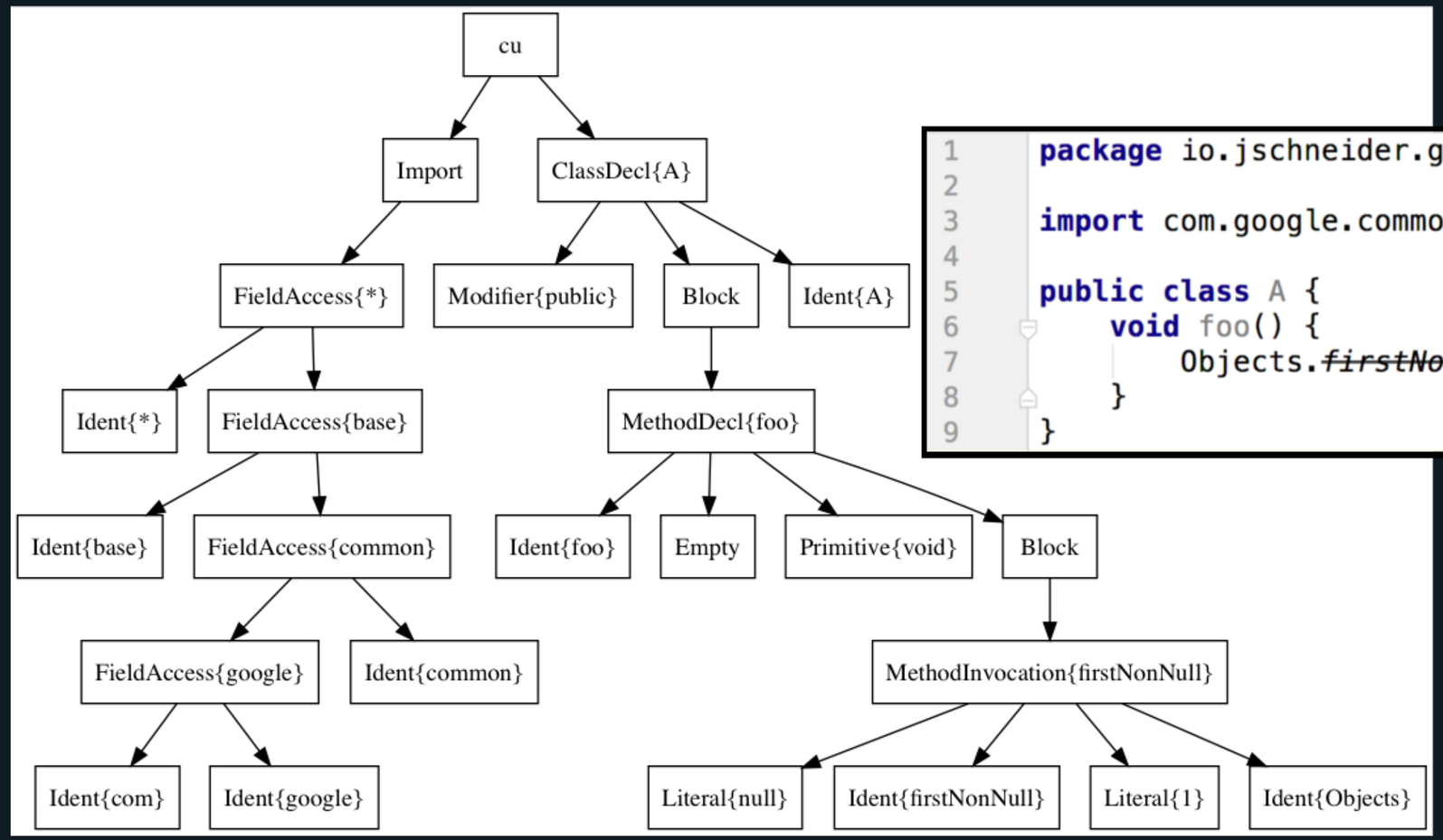
Raw source code + classpath = Rewrite AST.

```
String javaSource = /* Read A.java */;
List<Path> classpath = /* A list including Guava */;

Tr.CompilationUnit cu = new OracleJdkParser(classpath)
        .parse(javaSource);

assert(cu.firstClass().getSimpleName().equals("A"));
```

The Rewrite AST covers the whole Java language.




```
1  package io.jschneider.gradlesummit.rewrite;
2
3  import com.google.common.base.Objects;
4
5  public class A {
6      void foo() {
7          Objects.firstNonNull(null, 1);
8      }
9  }
```

Rewrite's AST is special.

1. Serializable
2. Acyclic
3. Type-attributed

Rewrite's AST preserves formatting.

```
public class A {
    public void foo() {
        List<String> l = Arrays.asList( ...a: "",
                "a",
                    "b",
                        "c"
        );
    }
}
```

```
List<String> l = Arrays.asList("",
        "a",
            "b",
                "c"
)
```

```
Tr.CompilationUnit cu = new OracleJdkParser().parse(aSource);
        assertThat(cu.print()).isEqualTo(aSource);

cu.firstClass().methods().get(0) // first method
  .getBody().getStatements() // method contents
  .forEach(t -> System.out.println(t.printTrimmed()));
```

We can find method calls and fields from the AST.

```
public class A {
    public void foo() {
        List<String> l = Arrays.asList(
                ...a: "a",
                "b",
                "c"
        );
    }
}
```

```
Tr.CompilationUnit cu = new OracleJdkParser().parse(aSource);

assertThat(cu.findMethodCalls("java.util.Arrays asList(..)")).hasSize(1);
assertThat(cu.firstClass().findFields("java.util.Arrays")).isEmpty();
```

We can find types from the AST.

```
public class A {
    public void foo() {
        List<String> l = Arrays.asList(
                ...a: "a",
                "b",
                "c"
        );
    }
}
```

```
assertThat(cu.hasType("java.util.Arrays")).isTrue();
assertThat(cu.hasType(Arrays.class)).isTrue();

assertThat(cu.findType(Arrays.class))
  .hasSize(1).hasOnlyElementsOfType(Tr.Ident.class);
```

Suppose we have a class referring to a deprecated Guava method.

```java
package io.jschneider.gradlesummit.rewrite;

import com.google.common.base.Objects;
import com.google.common.util.concurrent.MoreExecutors;

public class B {
    void foo() {
        Objects.firstNonNull(
                null,
                "hi"
        );

        MoreExecutors.sameThreadExecutor();
    }
}
```

We can refactor both deprecated references.

```
Tr.CompilationUnit cu = new OracleJdkParser().parse(bSource);
Refactor refactor = cu.refactor();

refactor.changeMethodTargetToStatic(
  cu.findMethodCalls("com.google..Objects firstNonNull(..)"),
  "com.google.common.base.MoreObjects"
);

refactor.changeMethodName(
  cu.findMethodCalls("com.google..MoreExecutors sameThreadExecutor()"),
  "directExecutor"
);
```

The fixed code emitted from Refactor can be used to overwrite the original source.

```java
import com.google.common.base.Objects;
import com.google.common.util.concurrent.MoreExecutors;

public class B {
    void foo() {
        Objects.firstNonNull(
                null,
                "hi"
        );

        MoreExecutors.directExecutor();
    }
}
```

```
// emits a string containing the fixed code, style preserved
refactor.fix().print();
```

refactor-guava contains all the rules for our Guava transformation.

```
refactor.changeMethodTargetToStatic(
        cu.findMethodCalls( signature: "com.google.common.base.Objects firstNonNull(..)"),
        toClass: "com.google.common.base.Objects"
);

refactor.changeMethodTargetToStatic(
        cu.findMethodCalls( signature: "com.google.common.collect.Iterators emptyIterator(..)"),
        toClass: "java.util.Collections"
);

refactor.changeMethodName(
        cu.findMethodCalls( signature: "com.google.common.util.concurrent.MoreExecutors sameThreadEx
        toName: "directExecutor"
);

refactor.changeMethodName(
        cu.findMethodCalls( signature: "com.google.common.util.concurrent.Futures get(java.util.conc
        toName: "getChecked"
);

refactor.changeMethodName(
        cu.findMethodCalls( signature: "com.google.common.util.concurrent.Futures transform(com.goog
```

Or we can emit a diff that can be used with `git apply`

```
// emits a String containing the diff
refactor.diff();
```

```
diff --git a/B.java b/B.java
index cf08ec7..14f2241 100644
--- a/B.java
+++ b/B.java
@@ -1,15 +1,15 @@
 package io.jschneider.gradlesummit.rewrite;

-import com.google.common.base.Objects;
+import com.google.common.base.MoreObjects;
 import com.google.common.util.concurrent.MoreExecutors;

 public class B {
     void foo() {
-        Objects.firstNonNull(
+        MoreObjects.firstNonNull(
                 null,
                 "hi"
         );

-        MoreExecutors.sameThreadExecutor();
+        MoreExecutors.directExecutor();
     }
 }
```

Intermezzo: The io.spring.rewrite plugin

Just annotate a static method to define a refactor rule.

```
@AutoRewrite(value = "reactor-mono-flatmap",
             description = "change flatMap to flatMapMany")
public static void migrateMonoFlatMap(Refactor refactor) {
  // a compilation unit for the source file we are refactoring
  Tr.CompilationUnit cu = refactor.getOriginal();

  refactor.changeMethodName(
    cu.findMethodCalls("reactor..Mono flatMap(..)"),
    "flatMapMany");
}
```

We have some handy Gradle tasks.

To generate a report of what should be refactored in your project based on the `@AutoRewrite` methods found, run:

```
./gradlew lintSource
```

To automatically fix your code (preserving all of your beautiful code style), run:

```
./gradlew fixSourceLint && git diff
```

Part 2: Using BigQuery to find all Guava code in Github

Identify all Java sources from BigQuery's Github copy.

```
SELECT *
FROM [bigquery-public-data:github_repos.files]
WHERE RIGHT(path, 5) = '.java'
```

Move Java source file contents to our dataset.

```
SELECT *
FROM [bigquery-public-data:github_repos.contents]
WHERE id IN (
  SELECT id
  FROM [myproject:gradle_summit.java_files]
)
```

Note: This will eat into your $300 credits.
It cost me ~$6 (1.94 TB).

Cut down the sources to just those that refer to Guava packages.

Getting cheaper now...

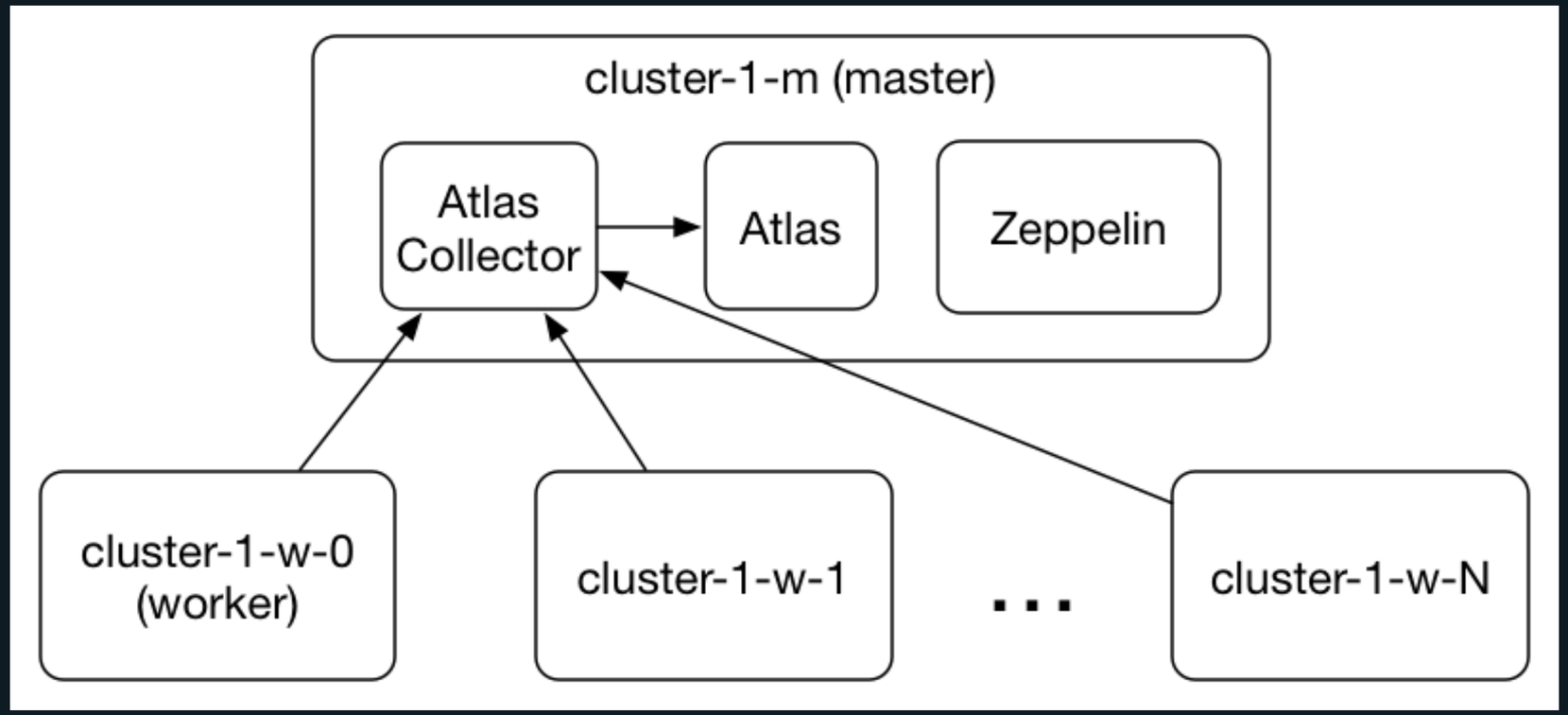
```
SELECT repo_name, path, content
FROM [myproject:gradle_summit.java_file_contents] contents
INNER JOIN [myproject:gradle_summit.java_files] files
  ON files.id = contents.id
WHERE content CONTAINS 'import com.google.common'
```

We now have the dataset to run our refactoring rule on.

1. 2.6 million Java source files.
2. 47,565 Github repositories.

Part 3: Employing our refactoring rule at scale on Google Cloud Dataproc.

Create a Spark/Zeppelin cluster on Google Cloud Dataproc.



Monitoring our Spark workers with Atlas and spring-metrics

```
@SpringBootApplication
@EnableAtlasMetrics // (1)
public class AtlasCollector {
    public static void main(String[] args) {
        SpringApplication.run(AtlasCollector.class, args);
    }
}
```

•••

Monitoring our Spark workers with Atlas and spring-metrics

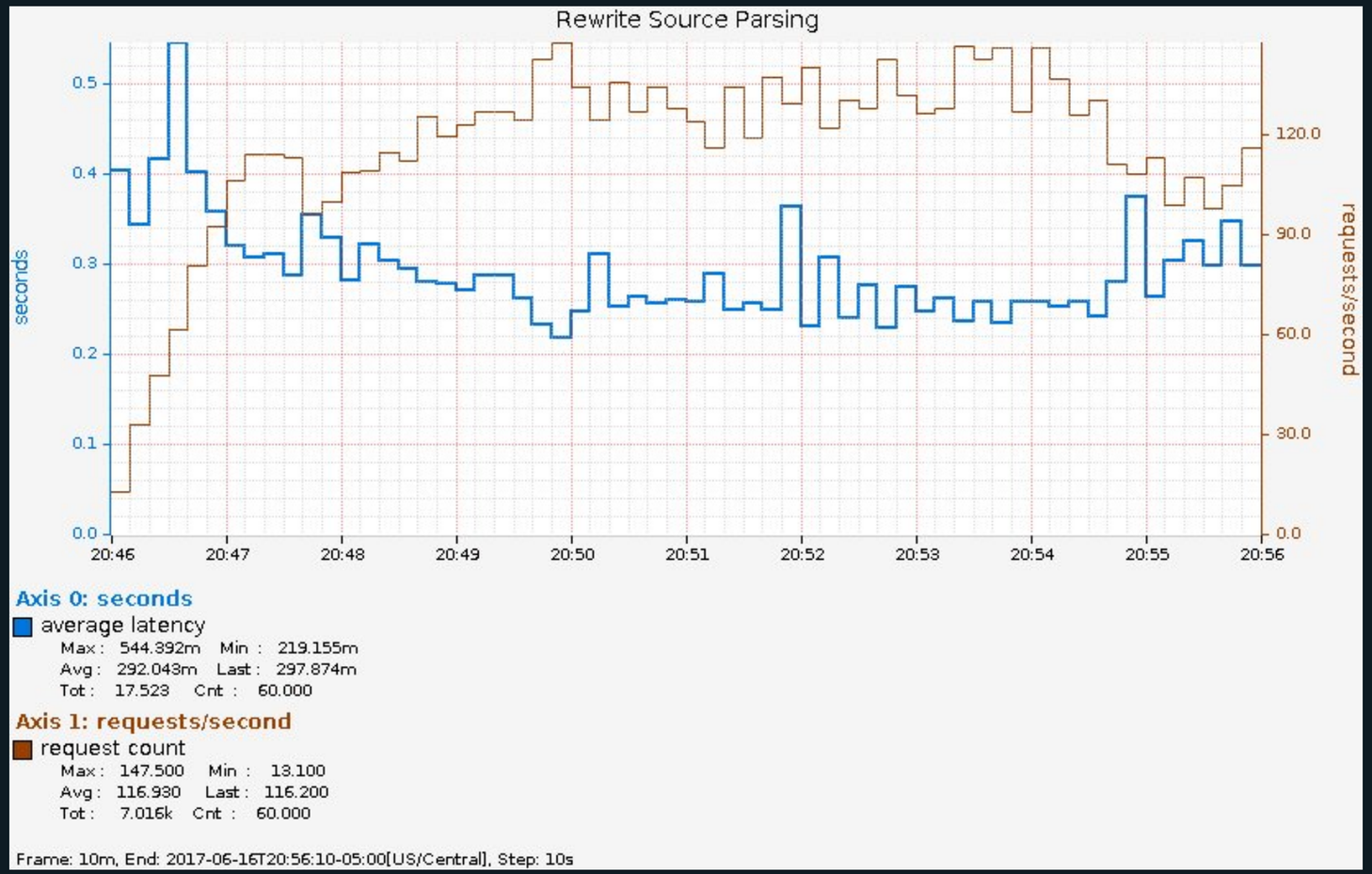
```
@RestController
@Timed // (2)
class TimerController {
  @Autowired MeterRegistry registry; // (3)
  Map<String, Timer> timers =
      new ConcurrentHashMap<>();

  @PostMapping("/api/timer/{name}/{timeNanos}")
  public void time(@PathVariable String name,
    @PathVariable Long timeNanos) {
      timers.computeIfAbsent(name, registry::timer)
          .record(timeNanos, TimeUnit.NANOSECONDS); // (4)
  }
}
```

We'll write the job in a Zeppelin notebook.

1. Select sources from BigQuery
2. Map over all the rows, parsing and running the refactor rule.
3. Export our results back to BigQuery.

Measuring our initial pass.



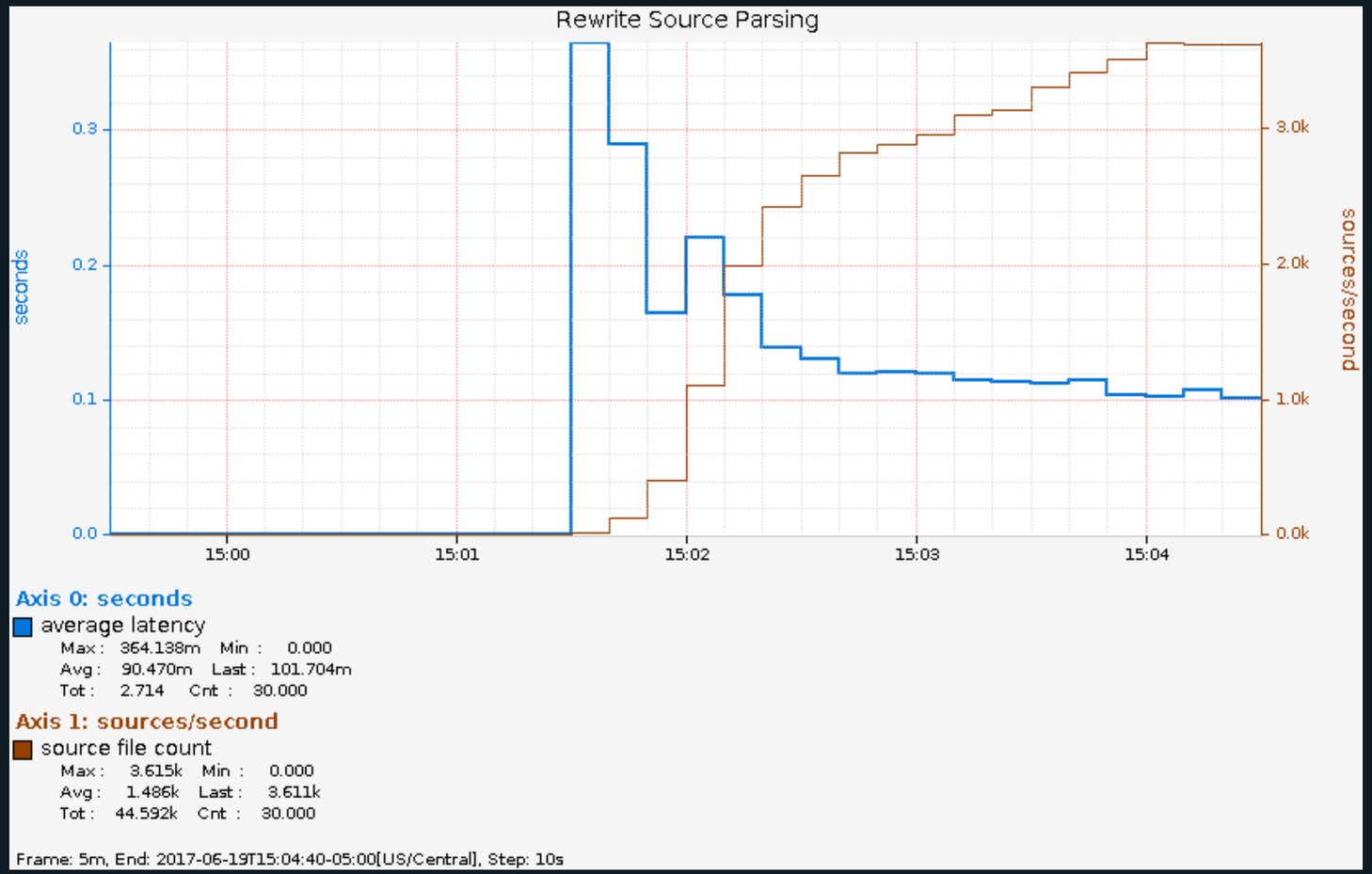
Measuring how big our cluster needs to be.

1. Rewrite averages 0.12s per Java source file
2. Rate of 6.25 sources per core / second
3. With 128 preemptible VMs, we've got:
   512 cores * 6.25 sources / core / second

3,200 sources / second = ~13 minutes total

We hope...

After scaling up the cluster with a bunch of cheap VMs.



Some source files are too badly formed to parse.

How many sources did we successfully parse?

FINISHED

```
refactored.count
```

```
res10: Long = 2590062
```

Took 10 sec. Last updated by anonymous at June 19 2017, 3:29:40 PM.

2,590,062/2,687,984 Java sources = 96.4%.

We found a healthy number of issues.

- 4,860 of 47,565 projects with problems
- 10.2% of projects with Guava references use deprecated API
- 42,794 source files with problems
- 70,641 lines of code affected

```
%sql
select issue, count(1) as total
from issues
group by issue
order by total
```

FINISHED

issue	total
Futures.get	291
TypeToken.isAssignableFrom	429
Futures.withFallback	445
Futures.transform	609
FutureFallback	724
OutputSupplier	1,157
MapConstraints	2,146
InputSupplier	2,681
Objects.firstNonNull	2,783
MoreExecutors.sameThreadExecutor	5,044
Iterators.emptyIterator	7,334
Objects.toStringHelper	23,155

github.com/jkschneider/gradle-summit-2017

Thanks for attending!