Factors Affecting Rule Performance

Charles Forgy

Production Systems Technologies, Inc.

July 2016

Objective

To demonstrate that there are some simple ways to diagnose and fix performance problems in forward-chaining rules.

Outline

- Background on Rule Engines.
- Useful Measurements.
- Diagnosing and Fixing Problems.

Background on Rule Engines

Example Rule

Applied when a customer checks out:

1.Gold rule: *A "Silver" customer with a shopping cart worth at least $2,000 is awarded the "Gold" status*.

Example Rule in OPSJ

```
rule gold
if {
    act:  Action(act.name == “CheckOut");
    cart: Cart(cart.amount >= 2000.00);
    cust: Customer(cust.name == act.customer;
                cust.cart == cart.customer;
                cust.level == “Silver”);
} do {
    update(cust).setLevel(“Gold");
}
```

Tokens

Rule engines keep information from cycle to cycle, updating the information as necessary to account for changes in the data.

A “token” is a list of one or more data objects that match one or more conditions.

- Conditions are processed in order, a token being created whenever a list of objects matching a list of conditions is found.
- Tokens are deleted when they are no longer valid.

Example Data for the Rule

- There is only one Action to check out.
- There are ten customers in the system that have carts containing over $2000.00 in goods. This includes the customer that is checking out.
- The customer that is checking out is a Silver customer.

The Tokens That Will Be Found.

- For act alone: one token.
- For act and cart together: ten tokens.
- For act, cart, and cust together: one token.

```
act:  Action(act.name == “CheckOut");
cart: Cart(cart.amount >= 2000.00);
cust: Customer(cust.name == act.customer;
           cust.name == cart.customer;
           cust.level == “Silver”);
```

After the Rule Fires

The action will change the customer level to “Gold”. Then the valid tokens will be:

- For act alone: one token.
- For act and cart together: ten tokens.
- For act, cart, and cust together: zero tokens.

```
act:  Action(act.name == “CheckOut");
cart: Cart(cart.amount >= 2000.00);
cust: Customer(cust.name == act.customer;
           cust.name == cart.customer;
           cust.level == “Silver”);
```

Useful Measurements

The Measurements

- How many tokens are compared at each condition.
- How many complete matches are found for the rule.

The engine can collect this information with little overhead.

WaltzDb Benchmark Program

- This is one of the most difficult of the standard benchmark programs.
- Just reading the rules will give very little insight into why the rules are expensive.

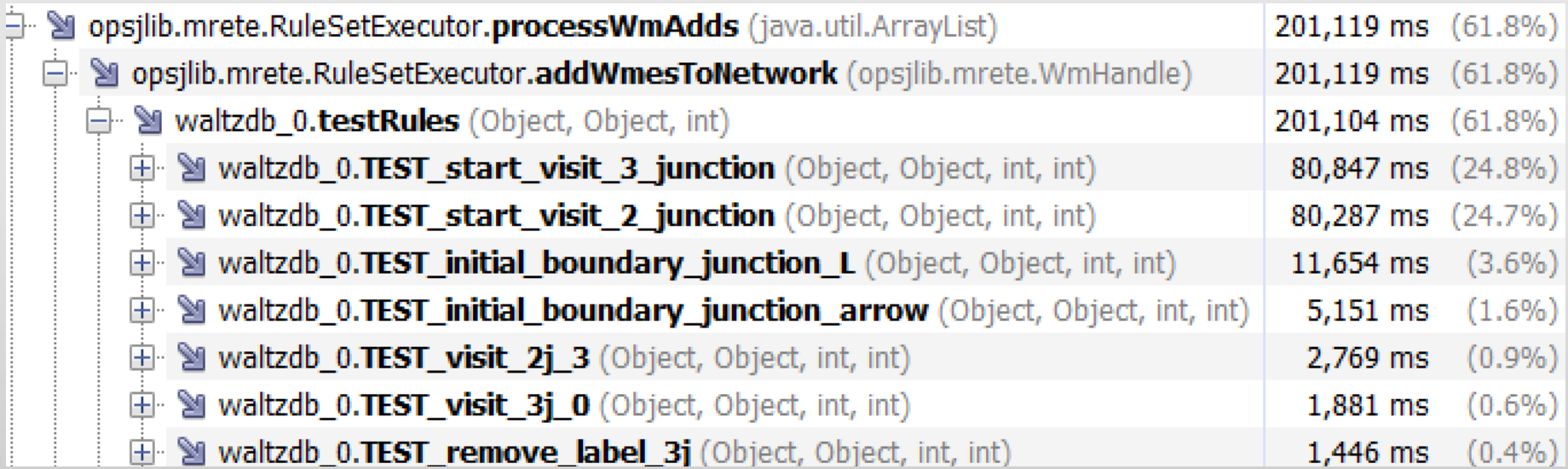
Five Highest Count Rules

Simply adding up all the token comparisons needed by each rule plus the number of matches found for that rule:

```
Rules Fired: 179072
Total time: 22.717 sec

waltzdb.start_visit_3_junction: 77514090
waltzdb.start_visit_2_junction: 67893332
waltzdb.initial_boundary_junction_L: 33890944
waltzdb.initial_boundary_junction_arrow: 14519436
waltzdb.checking: 725189
```

Confirmation From a Java Profiler



Diagnosing and Fixing Problems

The Worst Rule

```
rule start_visit_3_junction
if {
    stg:   stage(stg.value == "labeling");
    junct: junction(junct.kind == "3j",
                    junct.visited == "no");
} do {
    update(stg).setValue("visiting_3j");
    update(junct).setVisited("now");
}
```

More Detailed Measurements

```
Rules Fired: 179072
Total time: 22.602 sec

RULE start_visit_3_junction
Tokens compared at condition 2: 38757045
        Instantiations created: 38757045

RULE start_visit_2_junction
Tokens compared at condition 2: 33946666
        Instantiations created: 33946666
```

Throwing Away Information

```
rule start_visit_3_junction
if {
    stg:   stage(stg.value == "labeling");
    junct: junction(junct.kind == "3j",
                    junct.visited == "no");
} do {
    update(stg).setValue("visiting_3j");
    update(junct).setVisited("now");
}
```

Impact of the Update

```
Rules Fired: 179072
Total time: 22.602 sec

RULE start_visit_3_junction
Tokens compared at condition 2: 38757045
        Instantiations created: 38757045
                    Times fired: 6423

RULE start_visit_2_junction
Tokens compared at condition 2: 33946666
        Instantiations created: 33946666
                    Times fired: 5635
```

First Attempt to Fix the Rules

```
rule start_visit_3_junction
if {
    stg:   stage(stg.value == "labeling");
    junct: junction(junct.kind == "3j",
                    junct.visited == "no");
    !j2:   junction(j2.kind == "3j",
                    j2.visited == "no",
                    j2.base_point < junct.base_point);
} do {
    update(stg).setValue("visiting_3j");
    update(junct).setVisited("now");
}
```

Result

```
Rules Fired: 180467
Total time: 20.406 sec          (Before: 22.602)

RULE start_visit_3_junction
Tokens compared at condition 2: 56813011
Tokens compared at condition 3: 113820001
        Instantiations created: 12058
                   Times fired: 6423

RULE start_visit_2_junction
Tokens compared at condition 2: 15890700
Tokens compared at condition 3: 31820379
        Instantiations created: 5637
                   Times fired: 5635
```

Old vs New Results

```
RULE start_visit_3_junction
Tokens compared at condition 2: 38757045
        Instantiations created: 38757045

RULE start_visit_3_junction
Tokens compared at condition 2: 56813011
Tokens compared at condition 3: 113820001
        Instantiations created: 12058
                   Times fired: 6423
```

Second Attempt to Fix the Rules (By using MEA)

```
rule start_visit_3_junction
if {
    stg:   stage(stg.value == "labeling");
    junct: junction(junct.kind == "3j",
                    junct.visited == "no");
} do {
    // BEFORE: update(stg).setValue("visiting_3j");
    insert(new stage("visiting_3j"));
    update(junct).setVisited("now");
}
```

Also Required by MEA

```
rule stop_checking
if {
    stg:stage(stg.value == "checking");
} do {
    //OLD: update(stg).setValue("labeling");
    delete(stg);
}
```

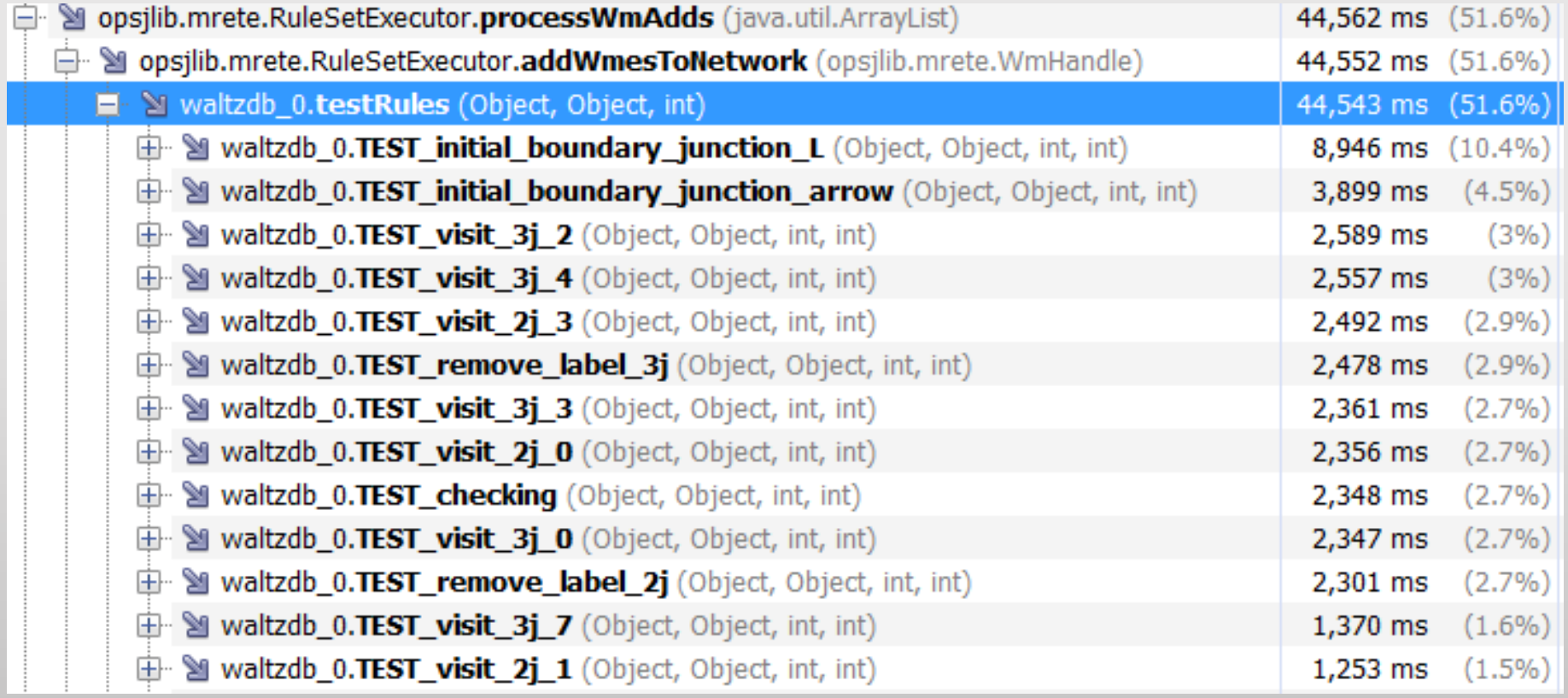
Result

```
Rules Fired: 179072
Total time: 8.815 sec          (Before: 22.602)

RULE start_visit_3_junction
Tokens compared at condition 2: 6423
        Instantiations created: 6423
                   Times fired: 6423

RULE start_visit_2_junction
Tokens compared at condition 2: 5635
        Instantiations created: 5635
                   Times fired: 5635
```

From the Java Profiler



All Results

```
RULE start_visit_3_junction
Tokens compared at condition 2: 38757045
        Instantiations created: 38757045
                    Times fired: 6423

RULE start_visit_3_junction
Tokens compared at condition 2: 56813011
Tokens compared at condition 3: 113820001
        Instantiations created: 12058
                    Times fired: 6423

RULE start_visit_3_junction
Tokens compared at condition 2: 6423
        Instantiations created: 6423
                    Times fired: 6423
```

Conclusions

- You can get a lot of information about performance of a rule by looking at just three factors:
  1. The number of tokens compared at each condition.
  2. The number of instantiations computed for the rule.
  3. The number of times the rule fires.
- This information does not correlate exactly with actual profiled times, but it is enough to identify many problematic rules.
- There are a number of ways to fix performance problems, and it may be necessary to try more than one.

Extra: What the Hardware is Doing

Waltzdb Analyzed

- Profiler: Intel VTune Amplifier.
- Processor: Intel “Skylake”.

Modern Processors

A Skylake core can:

- Decode and issue up to four instructions each clock.
- Retire up to four instructions each clock.
- To deal with dependencies and latencies, can have more than two hundred instructions “in-flight” (i.e., issued but not yet retired).

Overview

Elapsed Time: 222.718s

Clockticks:	1,121,043,681,563
Instructions Retired:	2,056,385,084,573
CPI Rate:	0.545
MUX Reliability:	0.989
Front-End Bound:	**9.9%**
Bad Speculation:	**6.3%**
Back-End Bound:	**36.3%**

Identify slots where no uOps are delivered due to a lack of required resources for accepting more uOps in the back-end of the pipeline. Back-end metrics describe a portion of the pipeline where the out-of-order scheduler dispatches ready uOps into their respective execution units, and, once completed, these uOps get retired according to program order. Stalls due to data-cache misses or stalls due to the overloaded divider unit are examples of back-end bound issues.

Retiring:	**47.5%**
Total Thread Count:	22
Paused Time:	0s

Details of the Back-End

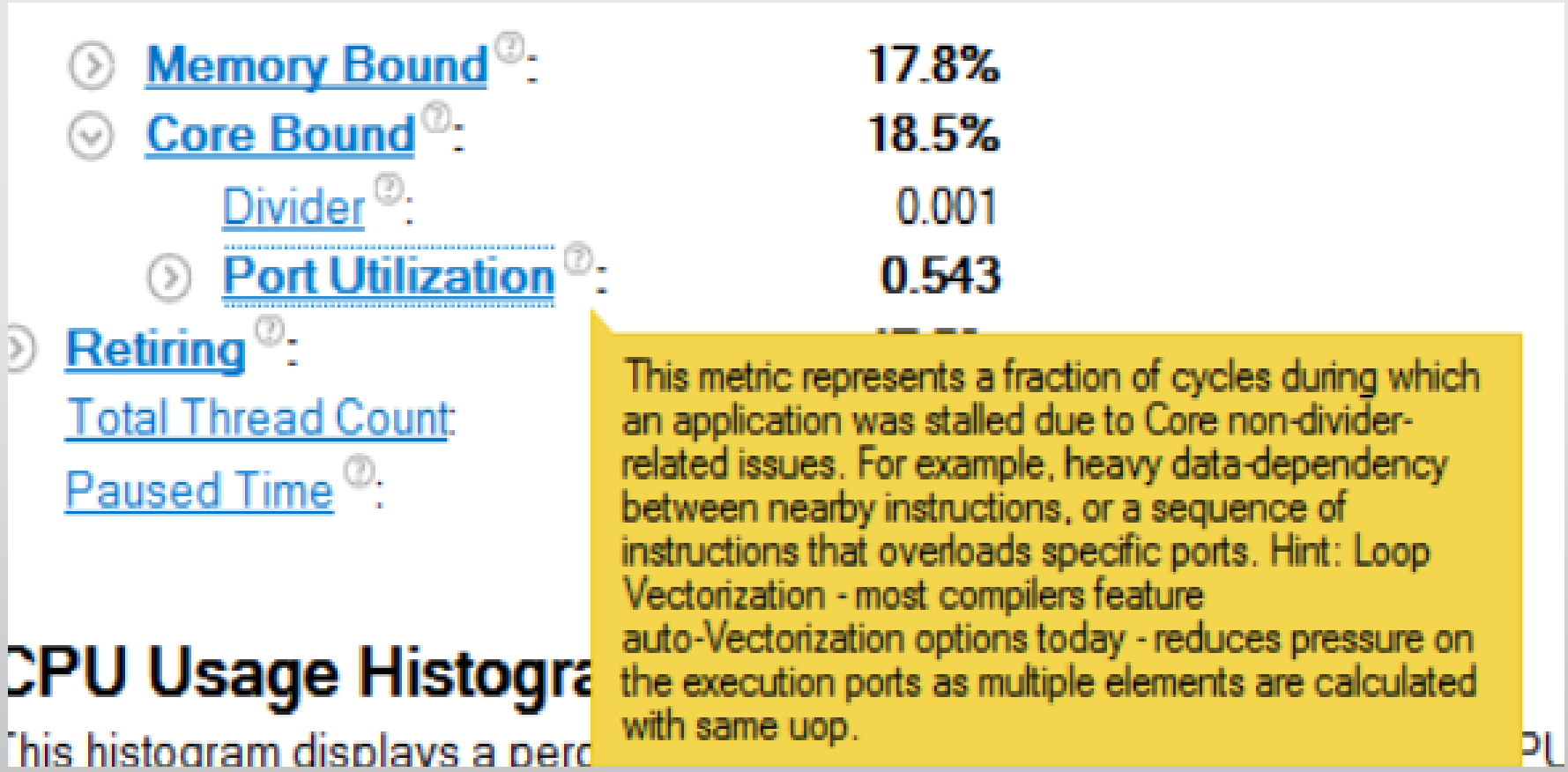
Elapsed Time: 222.718s

Clockticks:	1,121,043,681,563
Instructions Retired:	2,056,385,084,573
CPI Rate:	0.545
MUX Reliability:	0.989
Front-End Bound:	**9.9%**
Bad Speculation:	**6.3%**
Back-End Bound:	**36.3%**

Identify slots where no uOps are delivered due to a lack of required resources for accepting more uOps in the back-end of the pipeline. Back-end metrics describe a portion of the pipeline where the out-of-order scheduler dispatches ready uOps into their respective execution units, and, once completed, these uOps get retired according to program order. Stalls due to data-cache misses or stalls due to the overloaded divider unit are examples of back-end bound issues.

Memory Bound:	**17.8%**
Core Bound:	**18.5%**
Divider:	0.001
Port Utilization:	**0.543**
Retiring:	**47.5%**
Total Thread Count:	22
Paused Time:	0s

Why is it Core-Bound?



The Most Expensive Method

```
HashedAlphaTok findNextMatchTest()
{
    HashedAlphaTok atok;
    HashedAlphaTok next = cursor;

    while ((atok=next) != null) {
        Object aobj = atok.mObject;
        int hval = atok.mHashValue;
        next = atok.mFlink;
```

The Most Expensive Method

```
        try {
             if (hash==hval) {
                 pairscompared++;
                 if (testcls.zzeval(aobj, cevars, atok, btok)) {
                     cursor = next; return atok;
                 }
             }
         } catch (Exception ex) { engine.recordException(ex); }
    }           // End of while loop

    cursor = null; return null;
}
```

Overview in VTune

`while ((atok=next) != null) {`	
`Object aobj = atok.mObject;`	27,122,040,683
`int hval = atok.mHashValue;`	976,001,464
`next = atok.mFlink;`	1,954,002,931
`try {`	
`if (hash==hval) {`	6,284,009,426
`pairscompared++;`	4,514,006,771
`if (testcls.zzeval(aobj, cevars, atok, btok)) {`	2,230,003,345
`cursor = next; return atok;`	
`}`	
`}`	
`} catch (Exception ex) { engine.recordException(ex); }`	
`}  // End of while loop`	