TECHNISCHE UNIVERSITÄT DRESDEN

Department of Computer Science, Software Technology Group

On Performance of Delegation in Java.

Sebastian Götz (TU Dresden) and
Mario Pukall (University of Magdeburg)

HotSWUp 2009, Orlando, USA, 25.10.2009

Delegation

- One of the key mechanisms of object-orientation
- Heavily utilized in approaches for dynamic software upgrade
- Imposes performance penalties in terms of delayed execution
- But how big are these penalties ?

Experimental Setup

- Generation of classes, whose instances are connected to each other, forming a delegation chain

```
class C_i {
    private C_{i+1} next;
    public String call_i^1(t_1^1 p_1^1 ... t_j^1 p_j^1) {
        if(next != null) {
            double x = Math.sin(a) * ...
            return next.call_{i+1}^y (v_1^y, ..., v_k^y);
        } else return „end“;
    }
    ...
    public String call_i^k(t_1^k p_1^k ... t_q^k p_q^k) { ... }
    ...
}
```

workload

Experimental Setup

- Client, measuring the time **max** times (10.000) in order to take warm-up phase into consideration

```
class TestJIT {
   public static void main(String[] args) {
      //prepare
      for(int i = 0; i < max; i++) {
         long start = System.nanoTime();
         //invoke first method in chain
         long stop = System.nanoTime();
         //write result
      }
   }
}
```

Experimental Setup

- Generated client, measuring manually inlined methods

```
class TestManual {
   public void execute() {
      //insert all calculations
      long x_i = Math.sin(a_i) *
        Math.tan(b_i) + Math.hypot(c_i,d_i);
   }
   public static void main(String[] args) {
      TestManual test = new TestManual();
      for(int i = 0; i < max; i++) {
         long start = System.nanoTime();
         test.execute();
         long stop = System.nanoTime();
         //write results
      }
   }
}
```

Experimental Setup

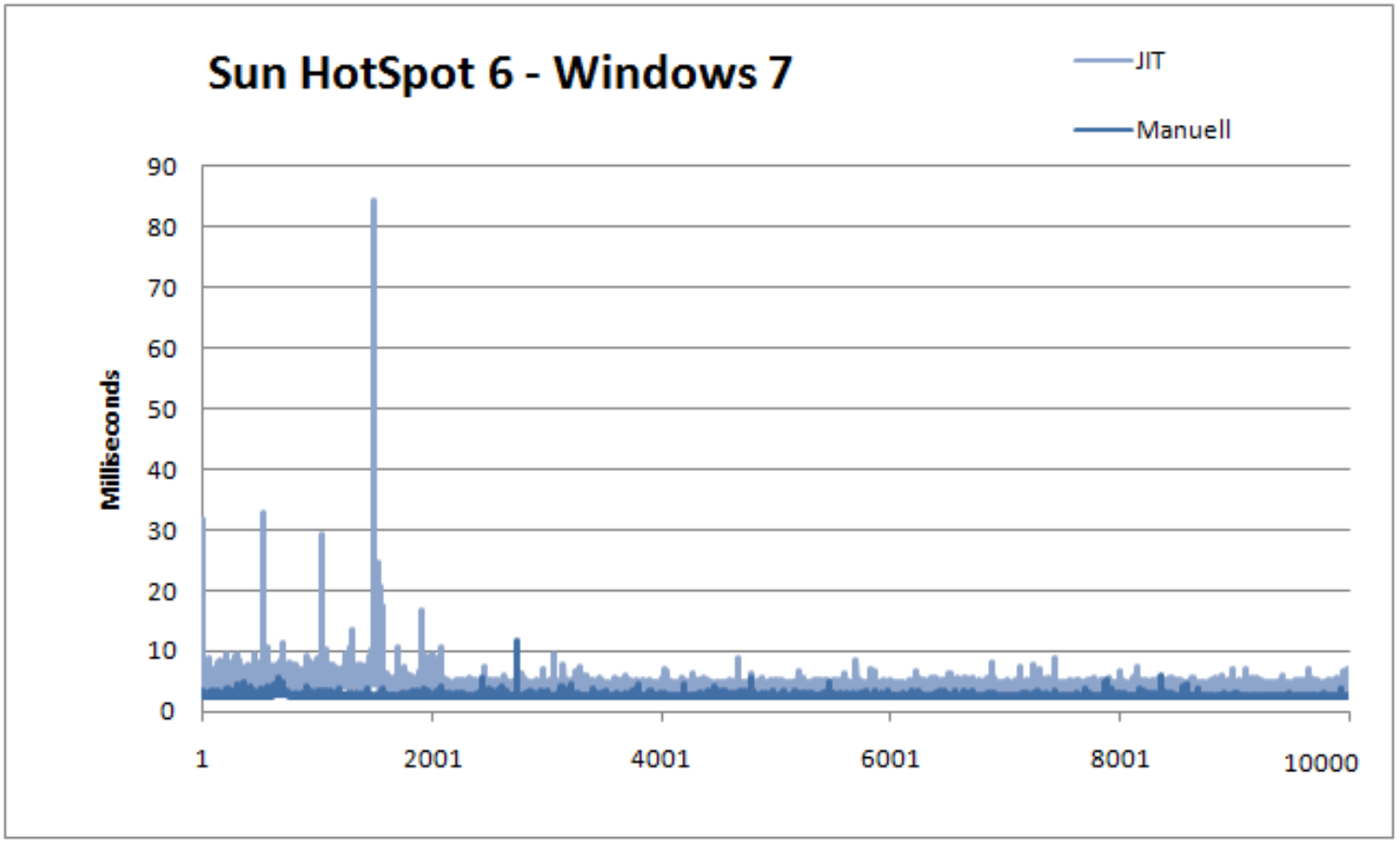
2 comparable **machines** used:

- Windows 7 and Linux (2.6.28) on
  - Intel Core 2 Duo T7700, 2.4GHz, 3GB RAM, 32bit
- Mac OS X on
  - MacBook Pro, Intel Core 2 Duo, 2.66GHz, 4GB RAM, 64bit

10 different **JVMs** observed:

- Sun HotSpot 5 + 6 for Windows 7 + Linux
- JRockit MC 3.1.0 for Java 5 + 6 for Windows 7 + Linux
- Apple HotSpot 5 + 6 for Mac OS X

Penalties are calculated using running average values.



Penalty

50%

But:

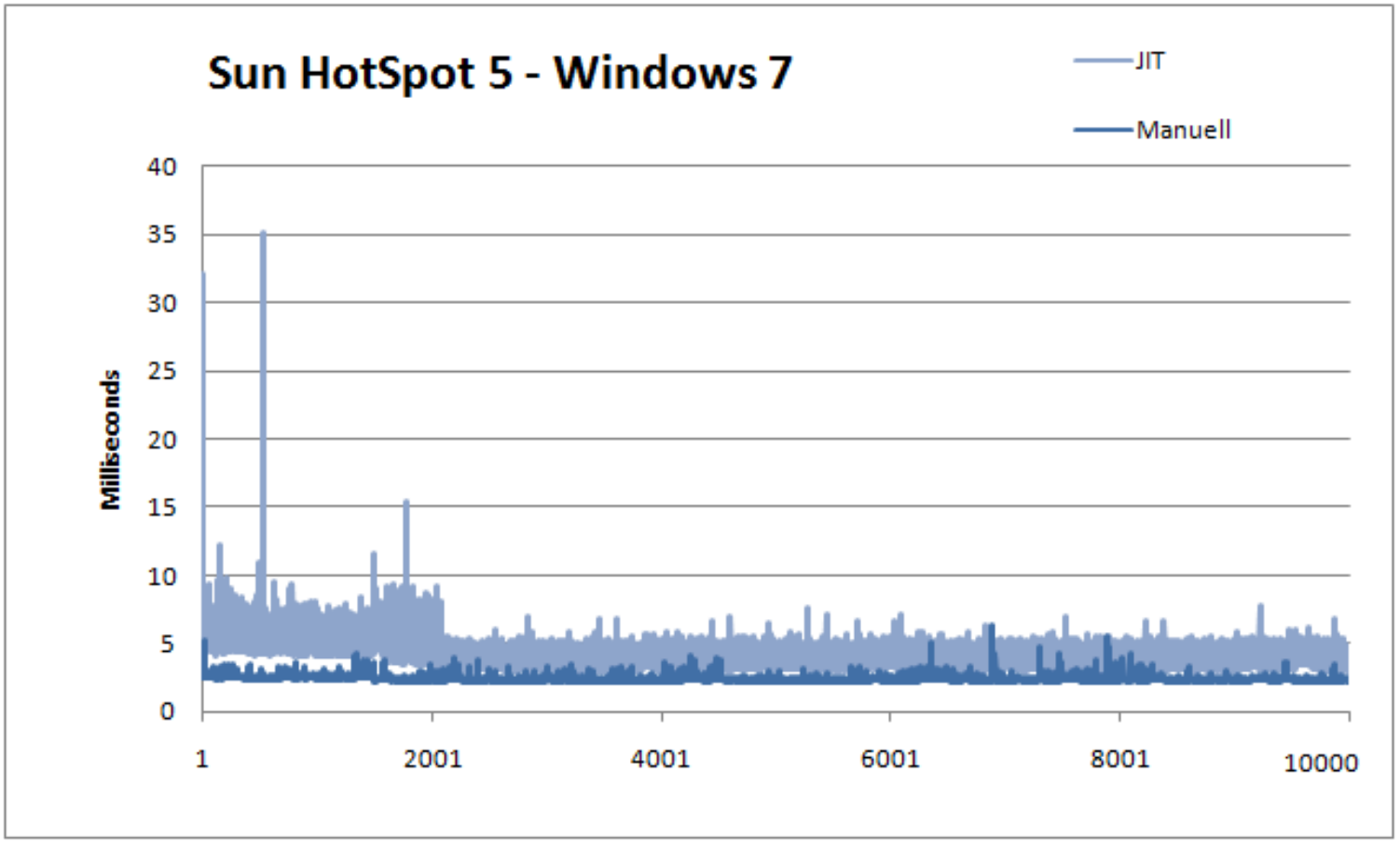
2x Workload:

32%

2,5x Workload:

12%

Windows 7



Penalty

46%

But:

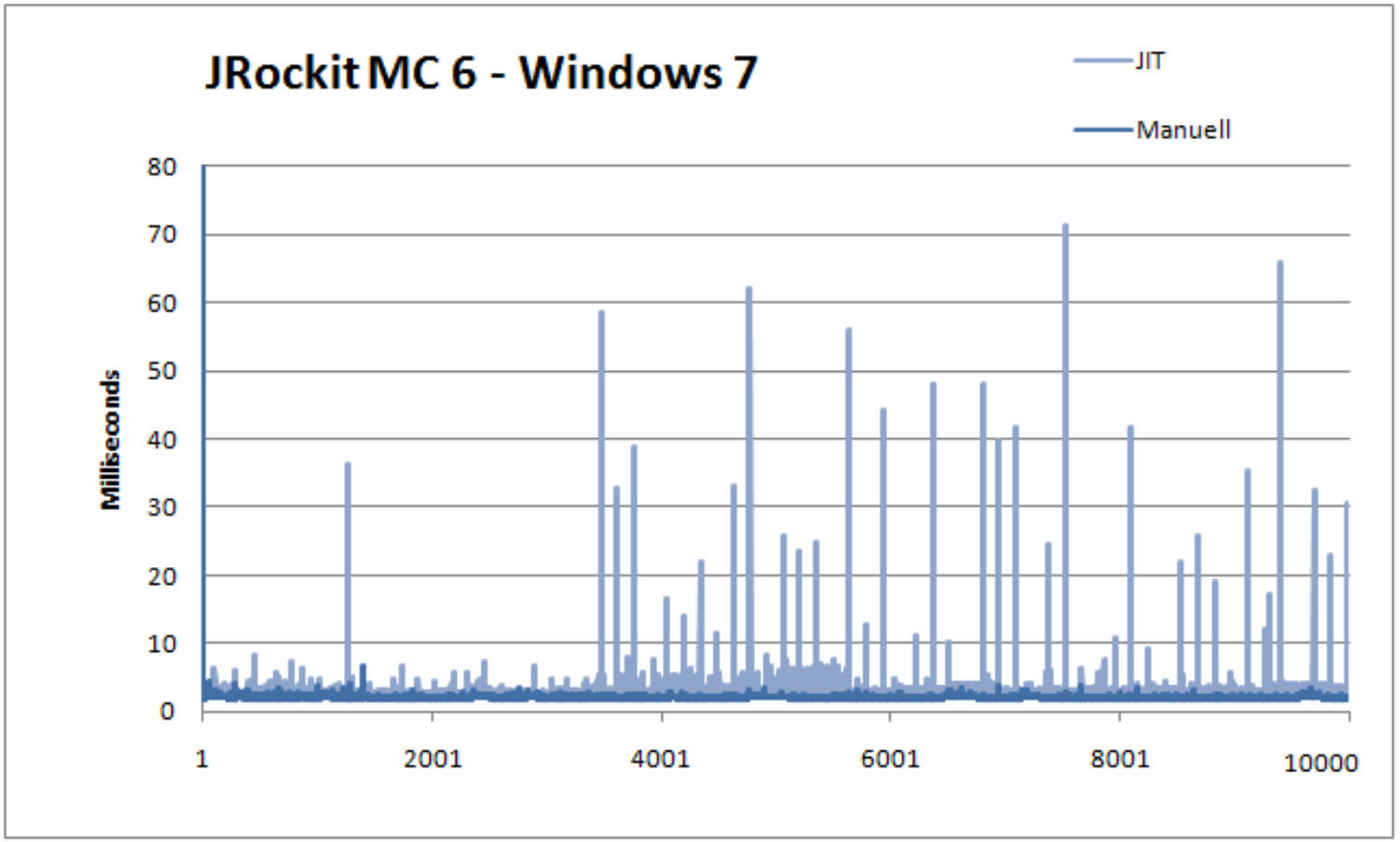
2x Workload:

14%

2,5x Workload:

14%

Windows 7



Penalty

43%

But:

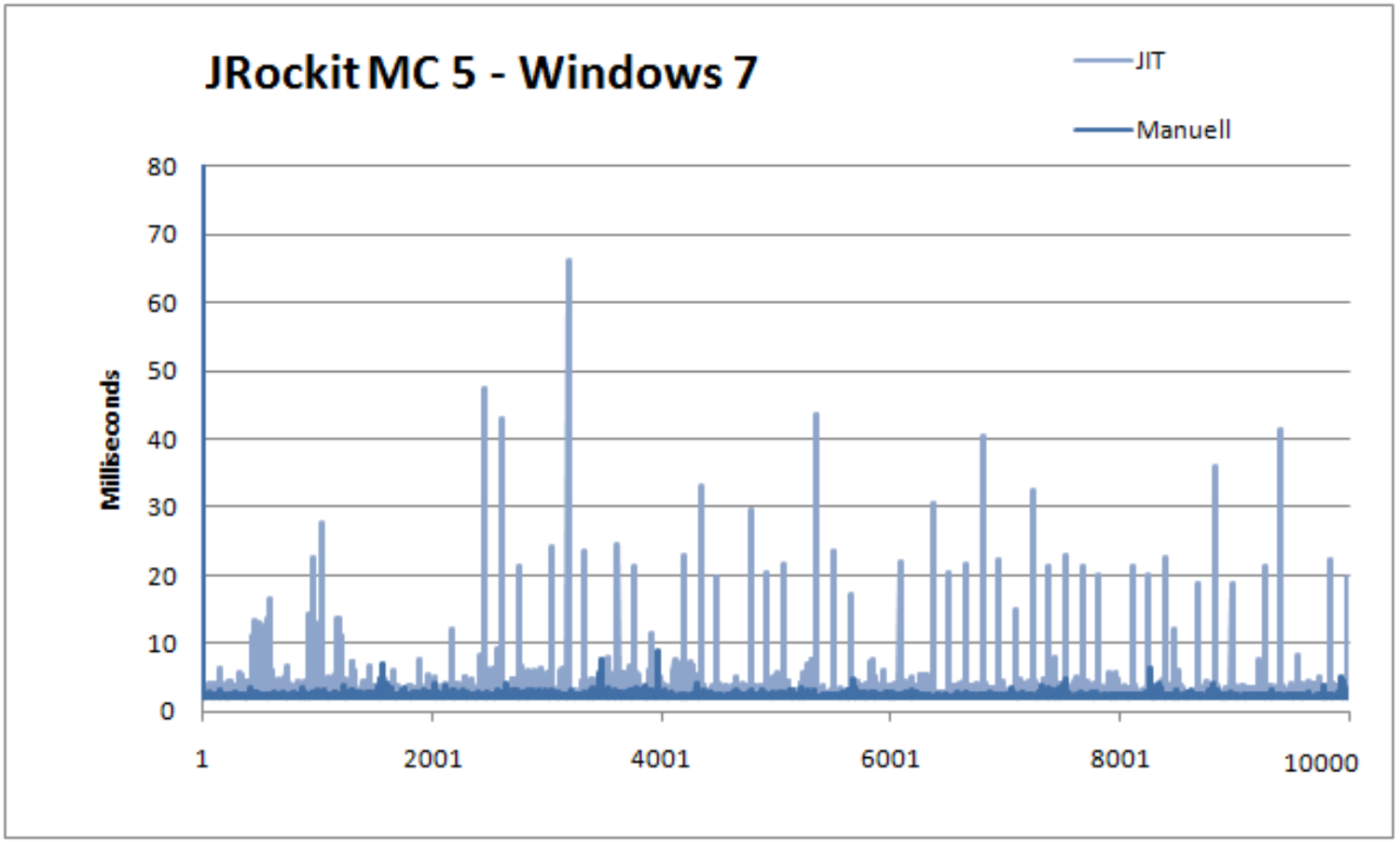
2x Workload:

31%

2,5x Workload:

2%

Windows 7



Penalty

39%

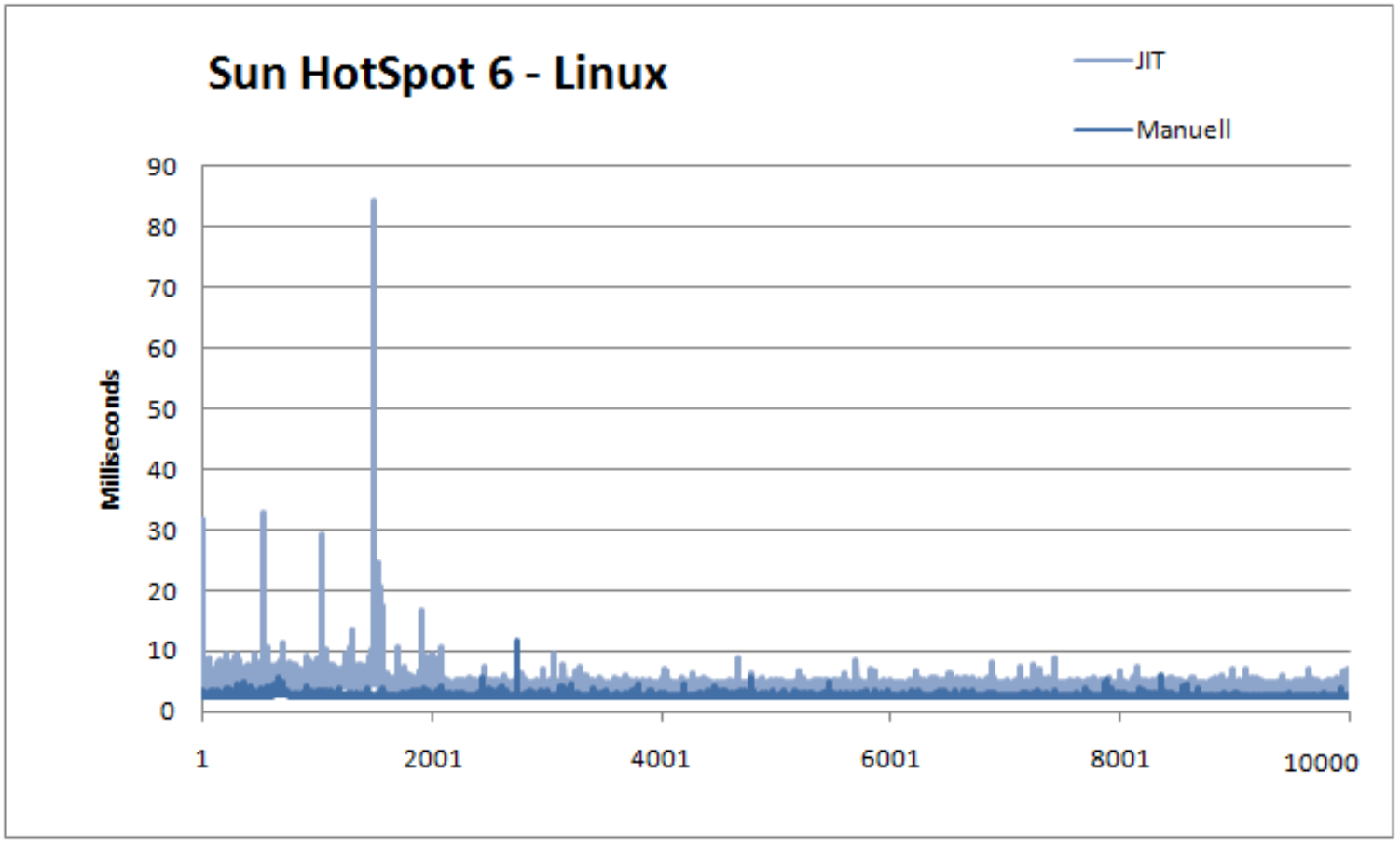
But:

2x Workload:

20%

2,5x Workload:

13%



Penalty

37%

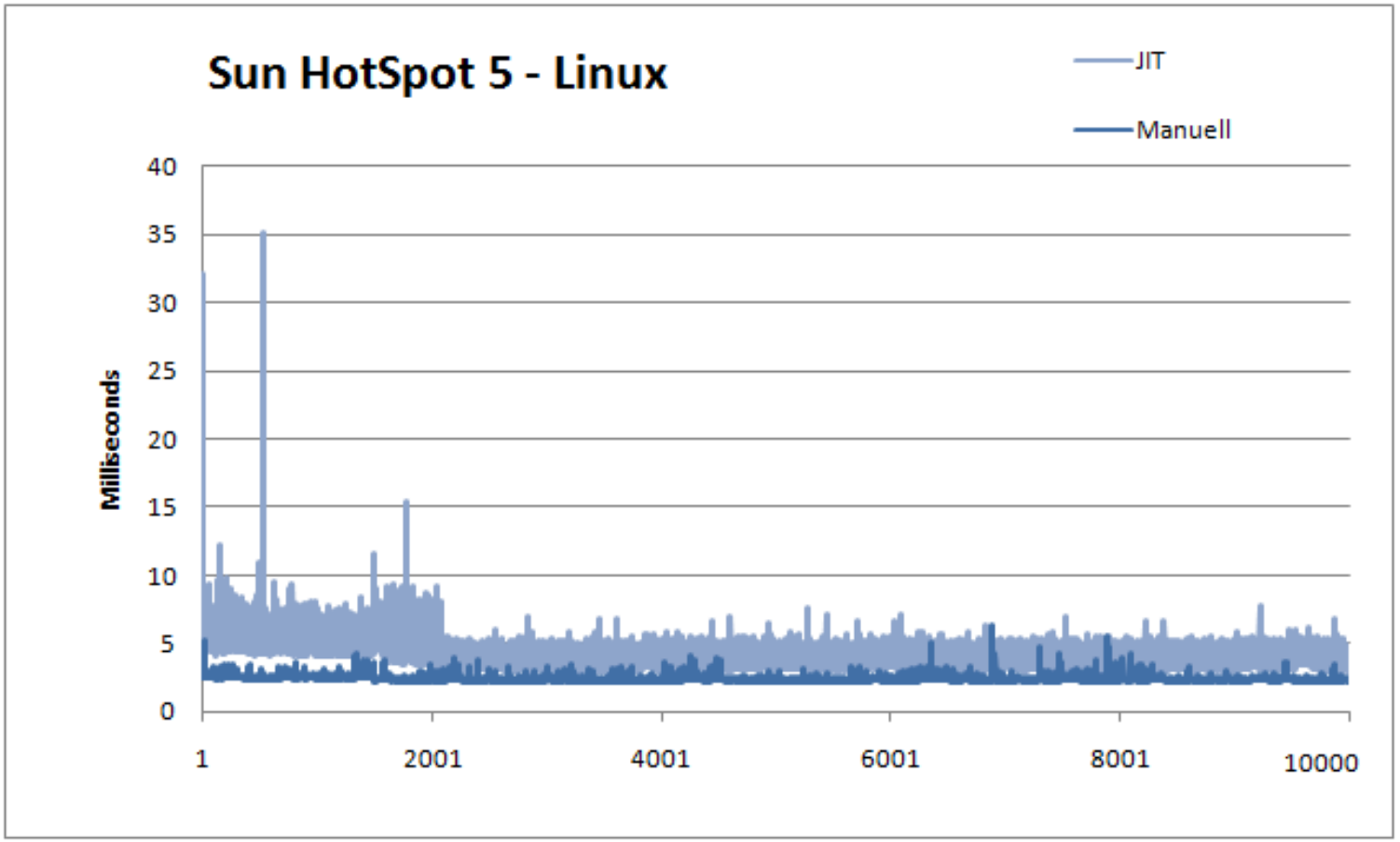
But:

2x Workload:

22%

2,5x Workload:

24%



Penalty

8%

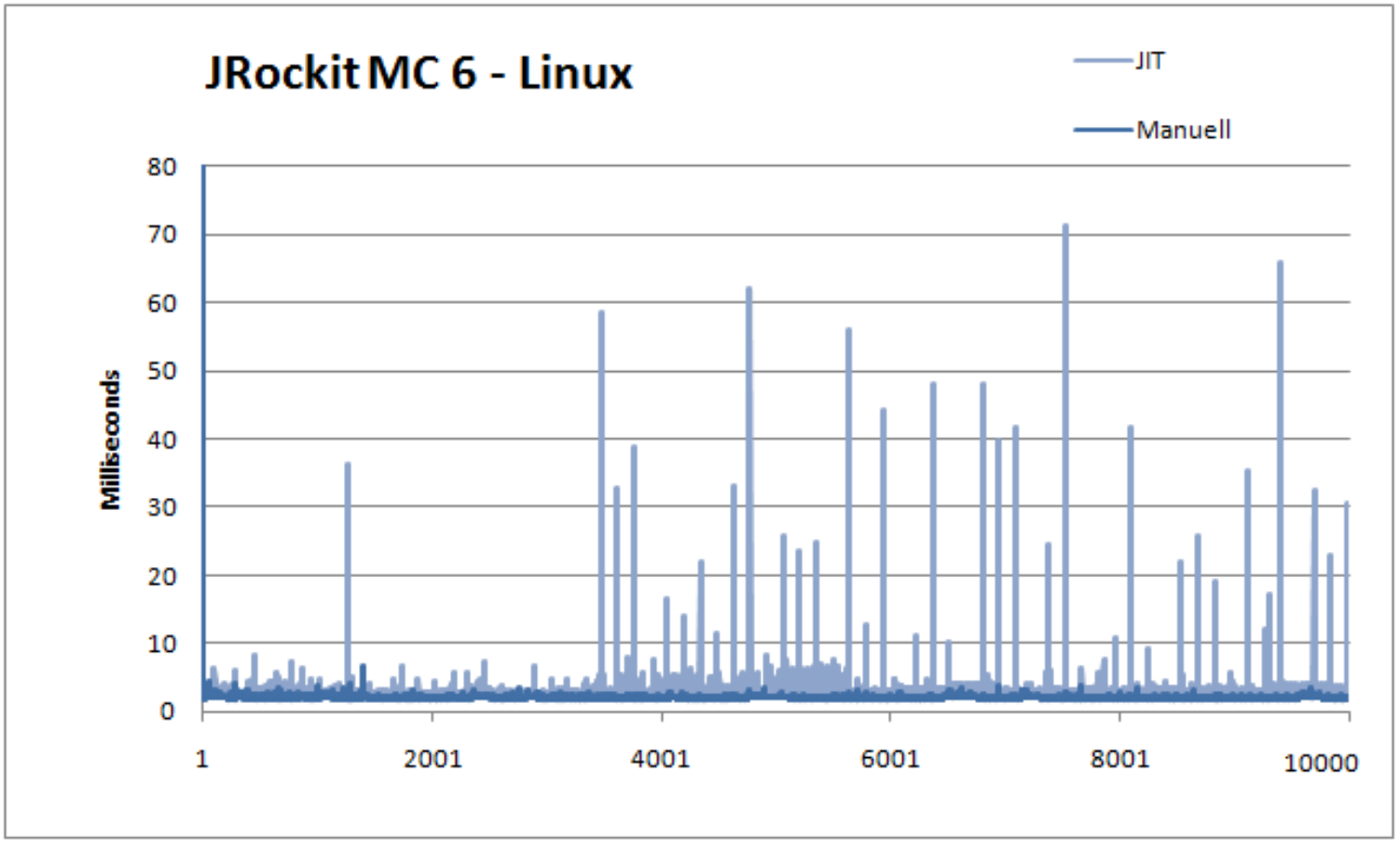
But:

2x Workload:

0%

2,5x Workload:

-8% !



Penalty

14%

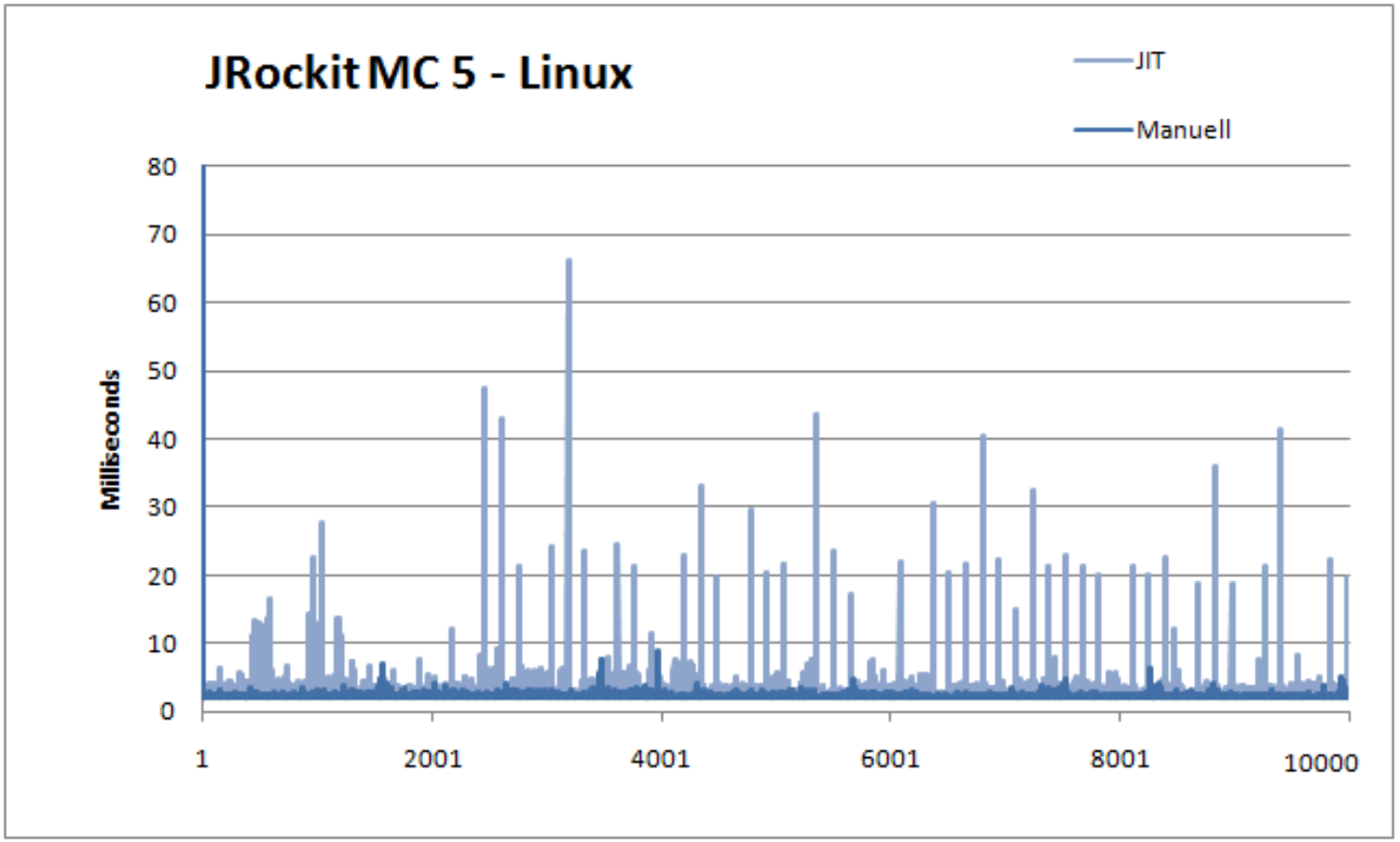
But:

2x Workload:

6%

2,5x Workload:

8%



Penalty

14%

But:

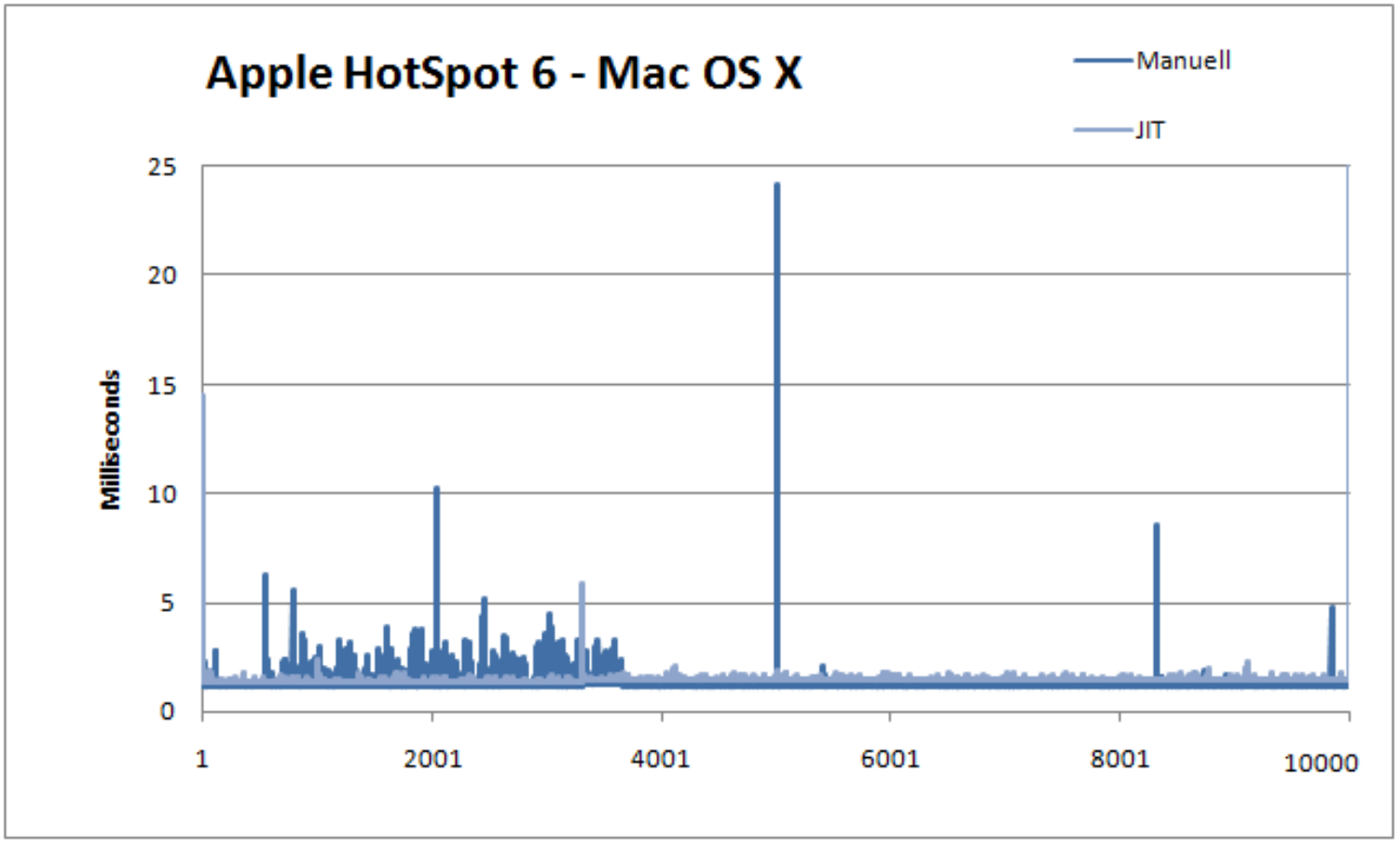
2x Workload:

13%

2,5x Workload:

7%

Mac OS X



Penalty

15%

But:

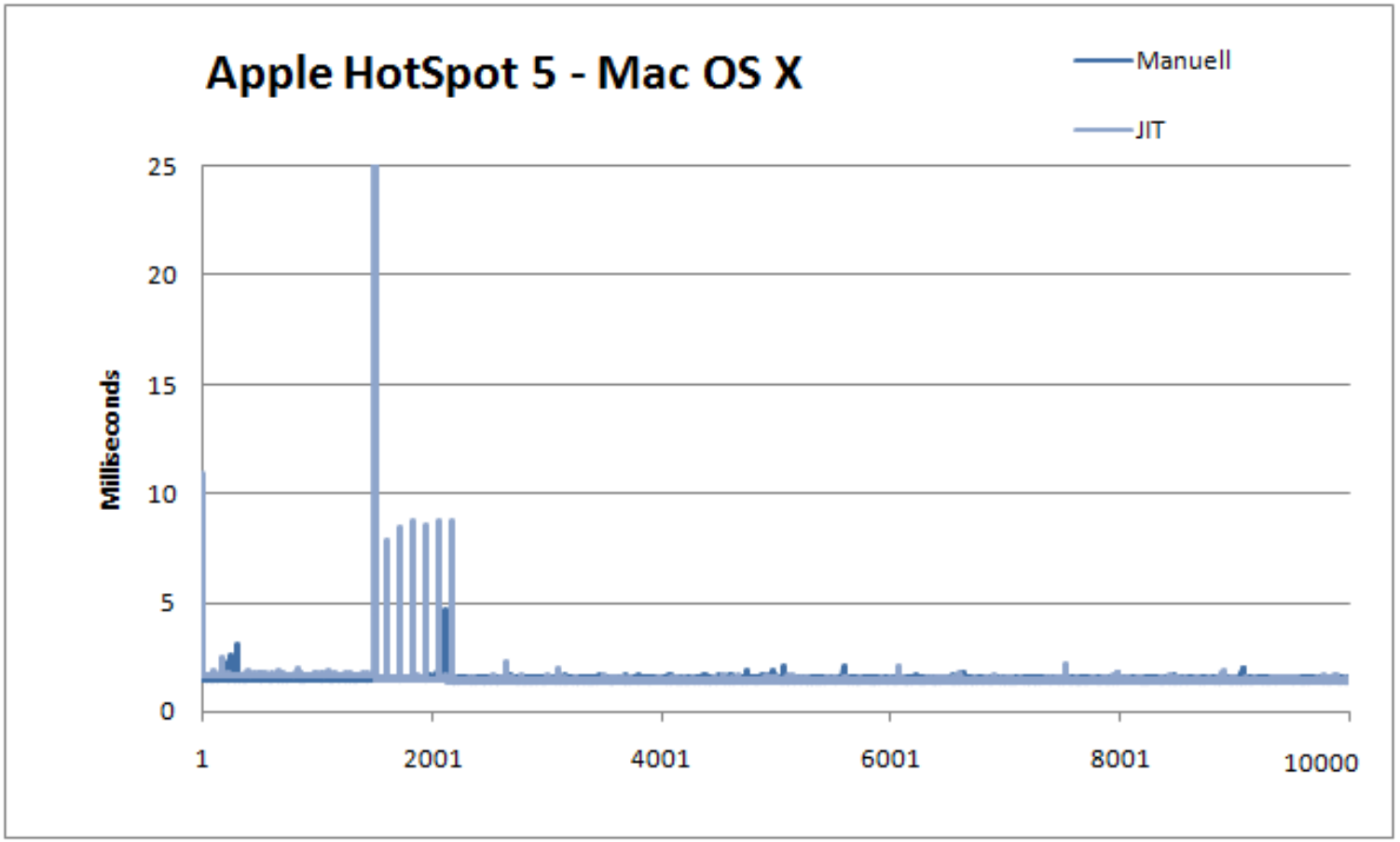
2x Workload:

7%

2,5x Workload:

6%

Mac OS X



Penalty

0%

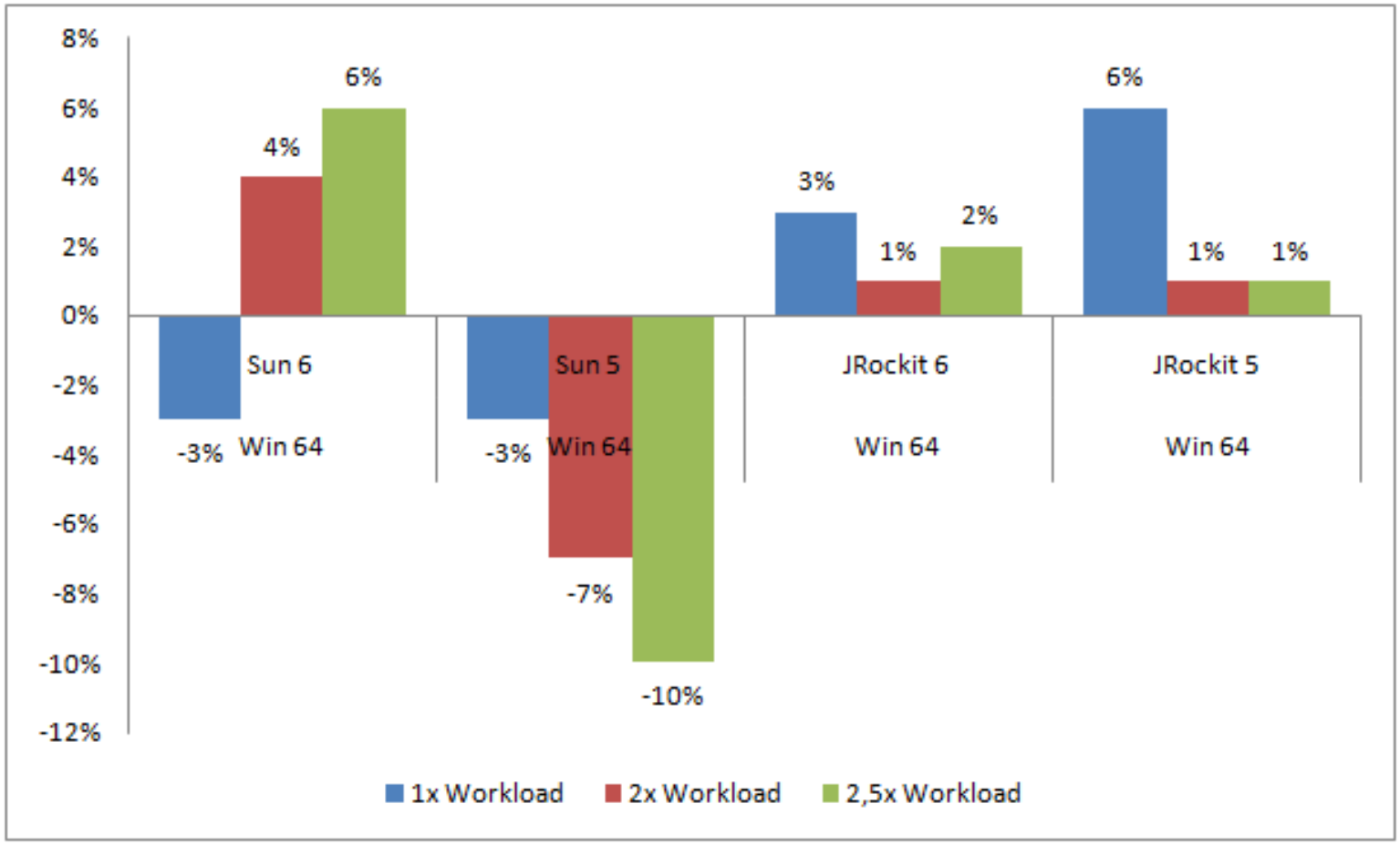
Possible Reason:
Highly optimized for architecture.

32bit vs. 64bit

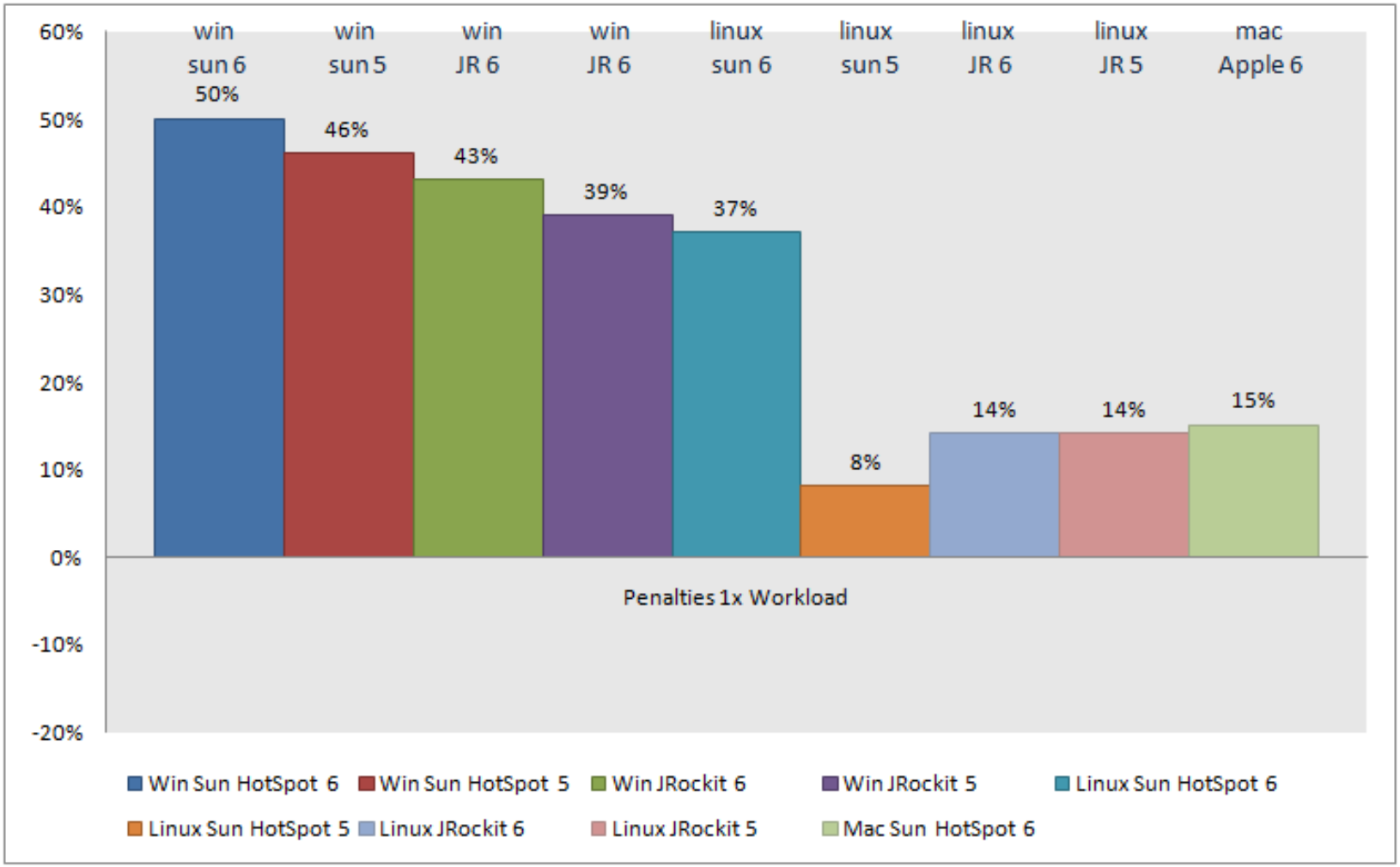
- Big difference between 32bit and **64bit** versions of the VMs
- Much better performance in 64bit VMs
- Sometimes Delegation is even faster, than manually inlined code

32bit vs. 64bit

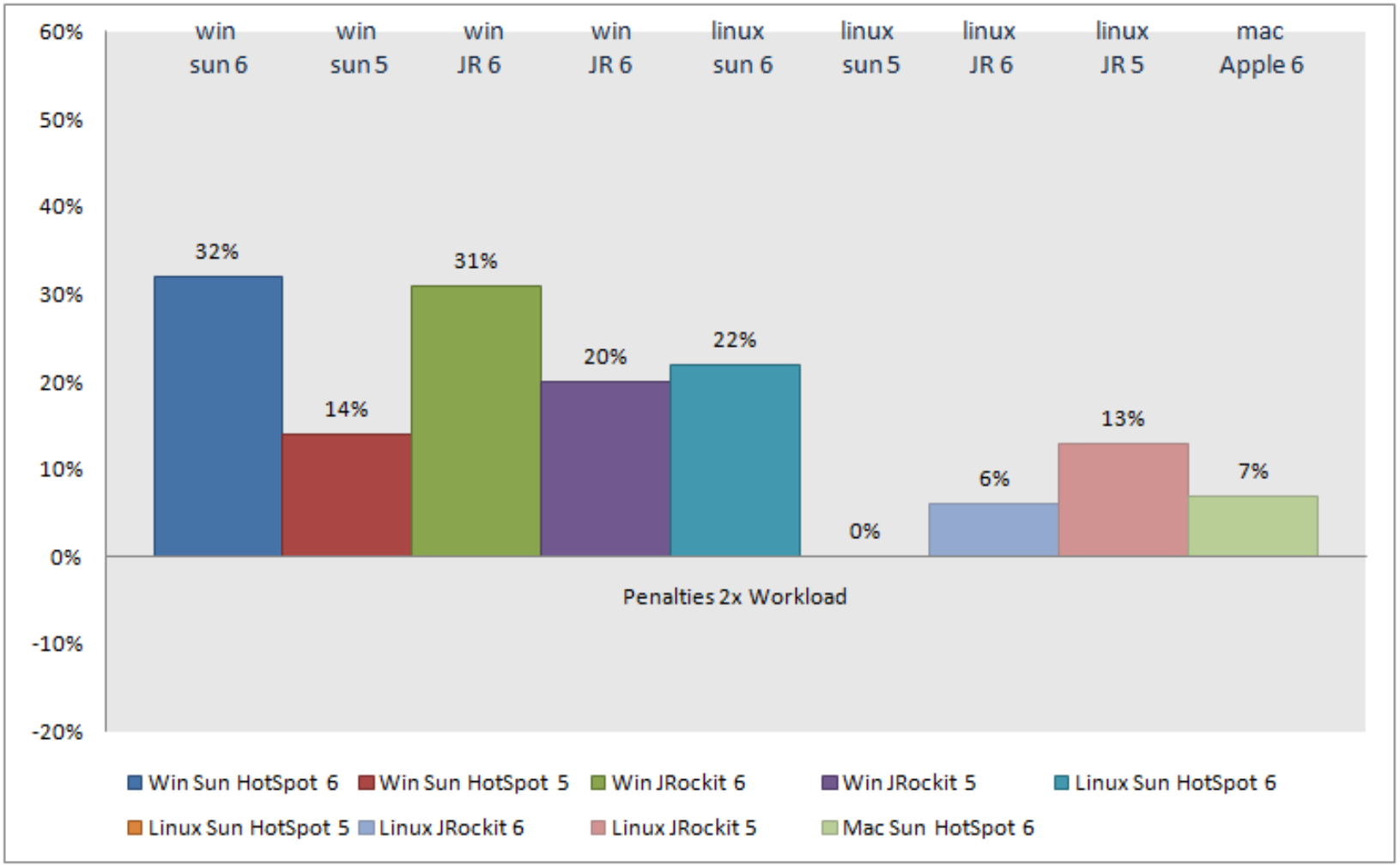
64bit Results for Windows 7, varying workload



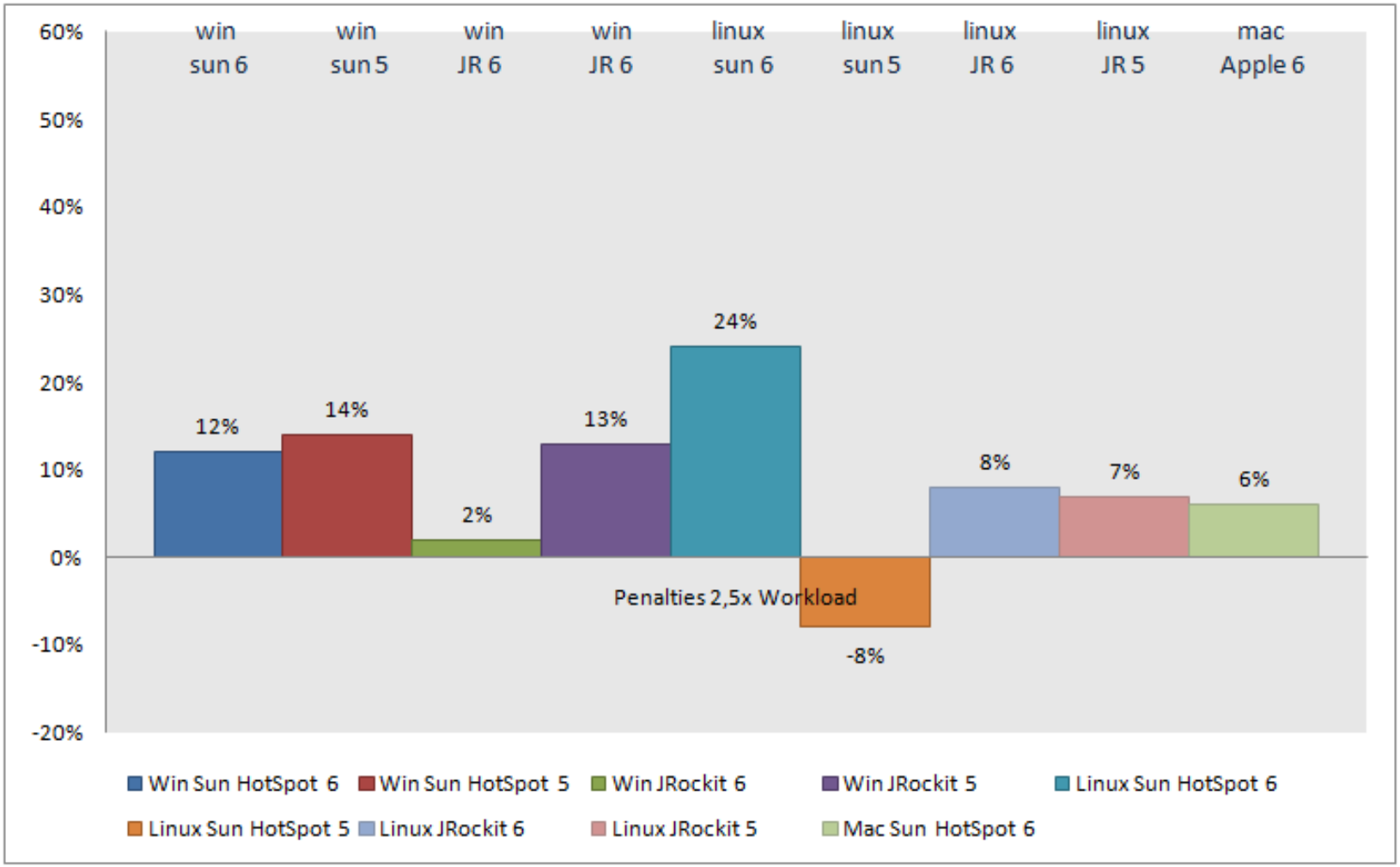
Conclusion


60%
50%
40%
30%
20%
10%
0%
-10%
-20%
win sun 6
win sun 5
win JR 6
win JR 6
linux sun 6
linux sun 5
linux JR 6
linux JR 5
mac Apple 6
50%
46%
43%
39%
37%
8%
14%
14%
15%
Penalties 1x Workload
Win Sun HotSpot 6
Win Sun HotSpot 5
Win JRockit 6
Win JRockit 5
Linux Sun HotSpot 6
Linux Sun HotSpot 5
Linux JRockit 6
Linux JRockit 5
Mac Sun HotSpot 6

Conclusion

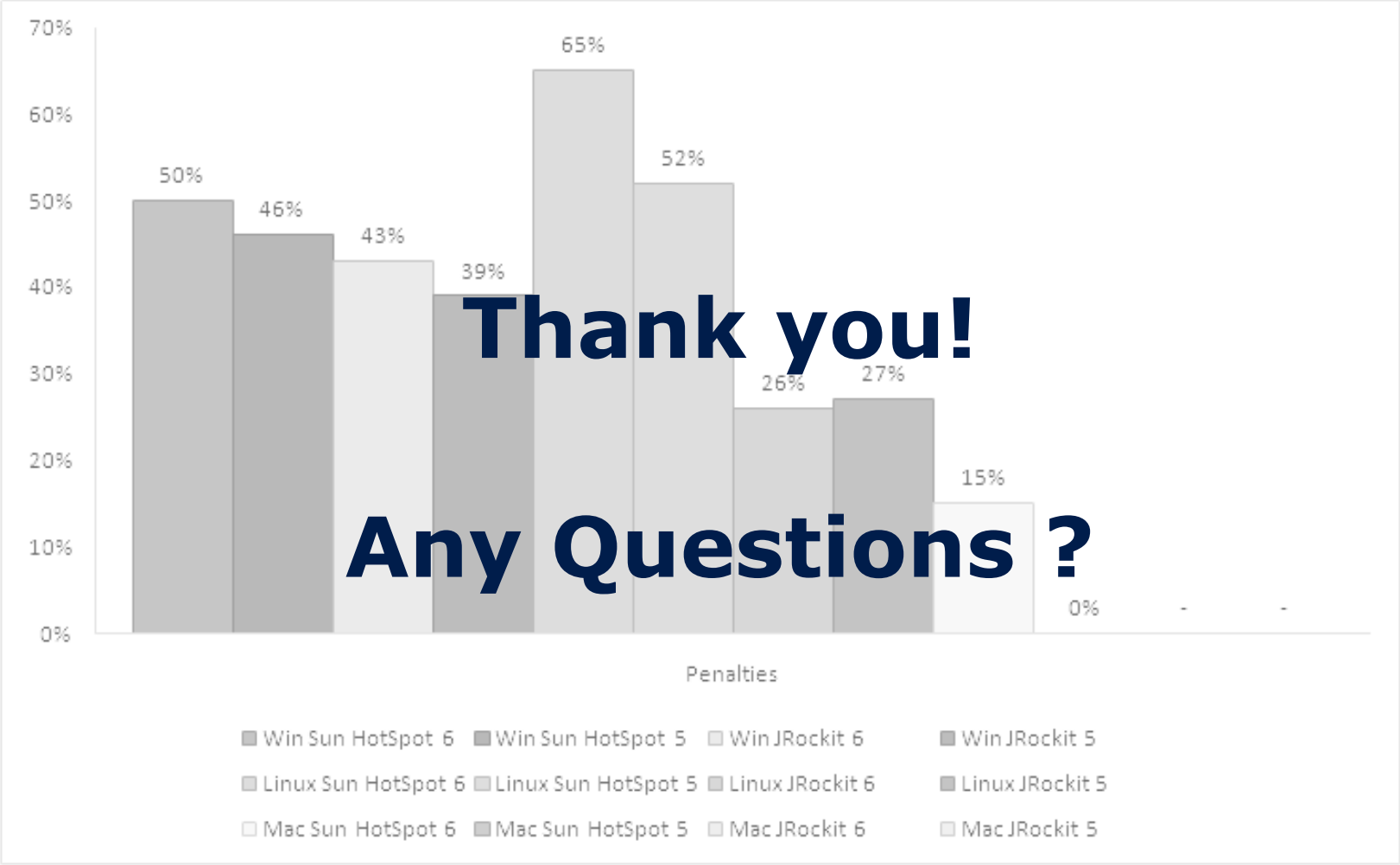

60%
50%
40%
30%
20%
10%
0%
-10%
-20%
win sun 6
win sun 5
win JR 6
win JR 6
linux sun 6
linux sun 5
linux JR 6
linux JR 5
mac Apple 6
32%
14%
31%
20%
22%
0%
6%
13%
7%
Penalties 2x Workload
Win Sun HotSpot 6
Win Sun HotSpot 5
Win JRockit 6
Win JRockit 5
Linux Sun HotSpot 6
Linux Sun HotSpot 5
Linux JRockit 6
Linux JRockit 5
Mac Sun HotSpot 6

Conclusion


60%
50%
40%
30%
20%
10%
0%
-10%
-20%
win sun 6
win sun 5
win JR 6
win JR 6
linux sun 6
linux sun 5
linux JR 6
linux JR 5
mac Apple 6
12%
14%
2%
13%
24%
-8%
8%
7%
6%
Penalties 2,5x Workload
Win Sun HotSpot 6
Win Sun HotSpot 5
Win JRockit 6
Win JRockit 5
Linux Sun HotSpot 6
Linux Sun HotSpot 5
Linux JRockit 6
Linux JRockit 5
Mac Sun HotSpot 6

Conclusion

- Valuable penalties (up to 50%)
- The more workload, the lower the penalty!
- 64bit JVMs give much better performance, than 32bit JVMs (at most **6%** penalty !)



Thank you!

Any Questions ?