Application Performance Monitoring: Trade-Off between Overhead Reduction and Maintainability

Jan Waller, Florian Fittkau, and Wilhelm Hasselbring

2014-11-27

1. Introduction

2. Foundation

3. Performance Benchmark

4. Overhead Reduction and its Impact on Maintainability

5. Related Work

6. Future Work and Conclusions

7. References

- Application level monitoring introduces monitoring overhead
- Live trace processing approaches rely on high throughput
- How to achieve?

- Application level monitoring introduces monitoring overhead
- Live trace processing approaches rely on high throughput
- How to achieve?

→ Structured process for performance tunings utilizing benchmarks

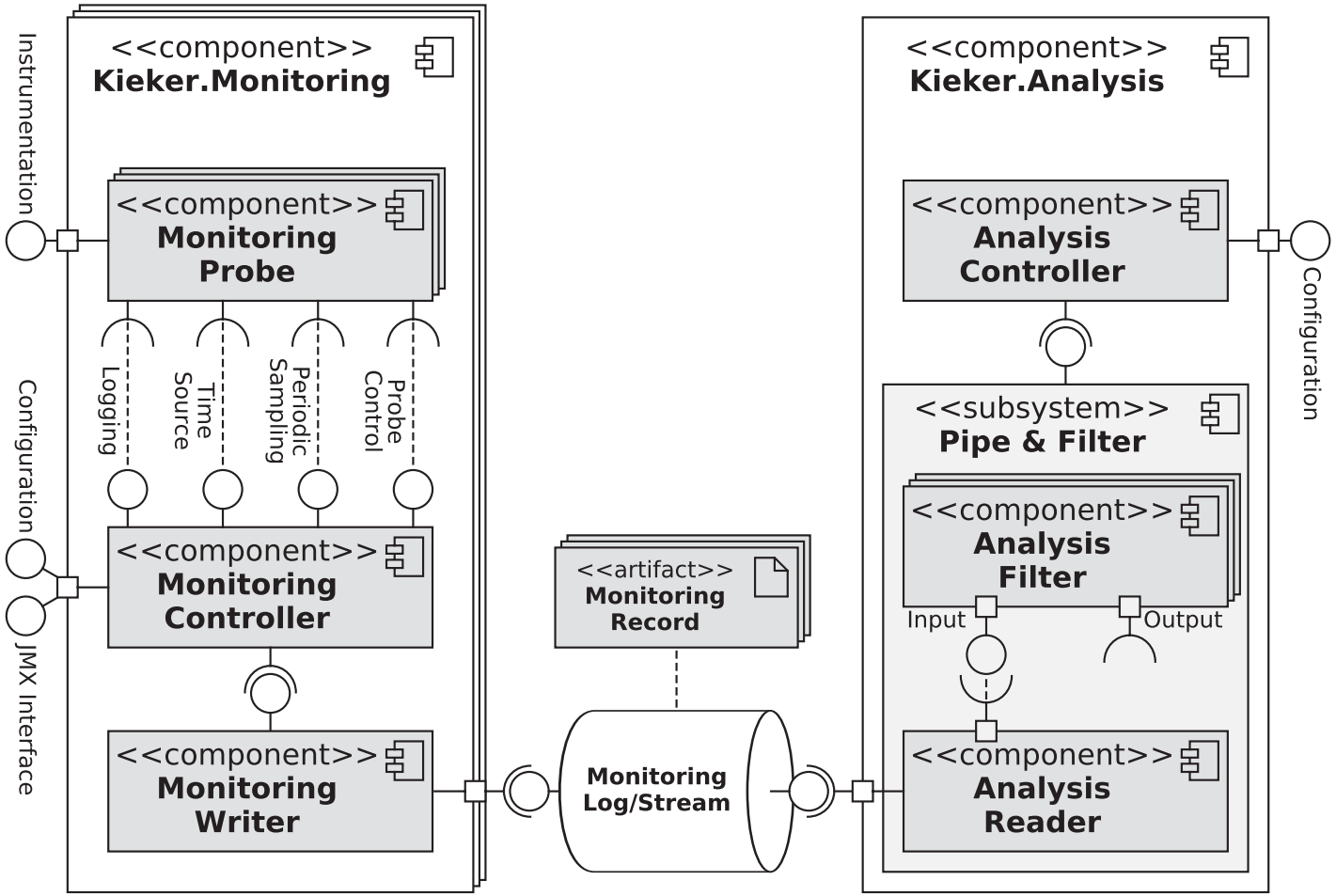

Figure 1: UML component diagram of a top-level view on the Kieker framework architecture

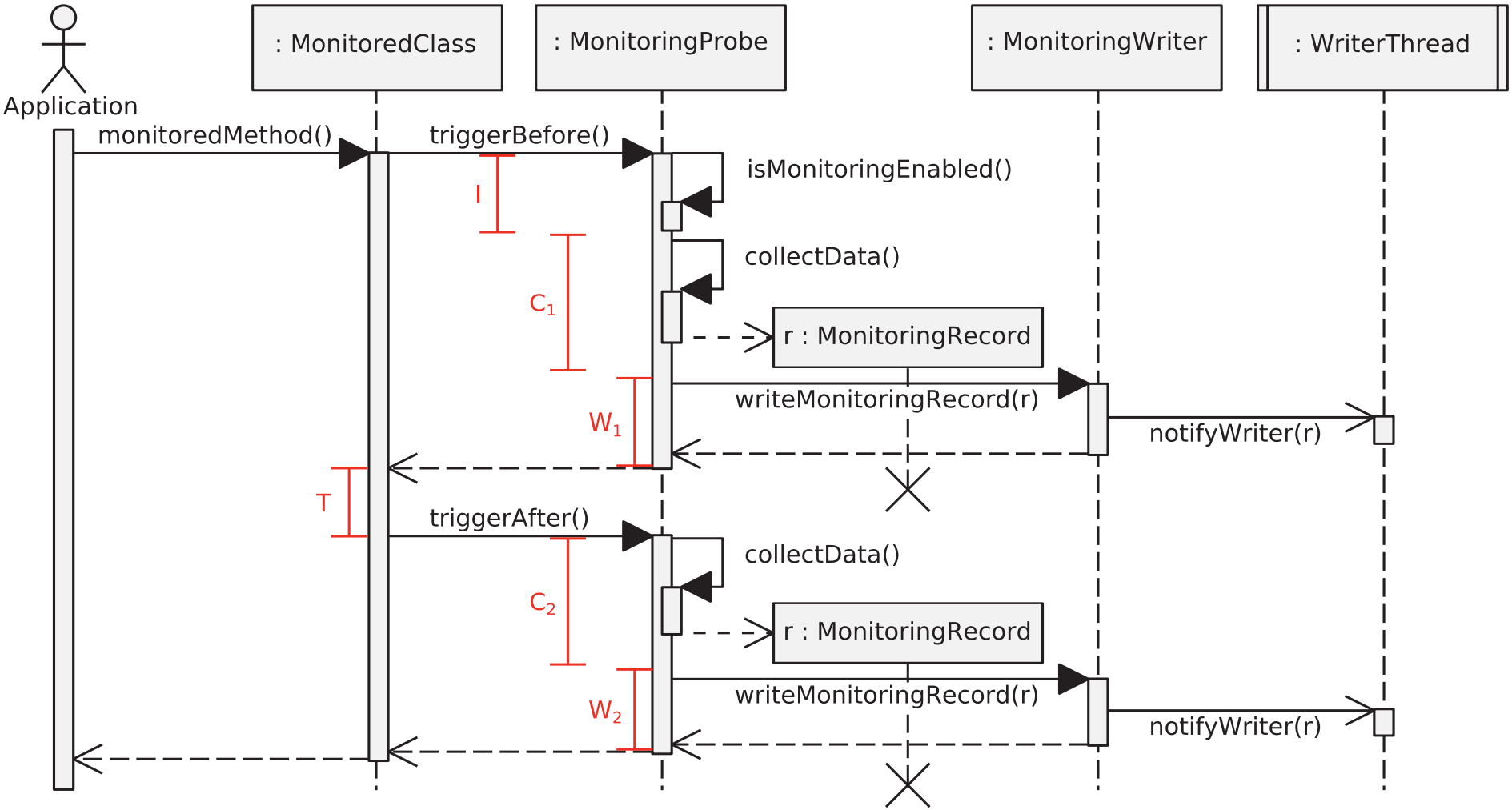

Figure 2: UML sequence diagram for method monitoring with the Kieker framework [WH13]

Figure 3: Benchmark engineering phases [WH13]

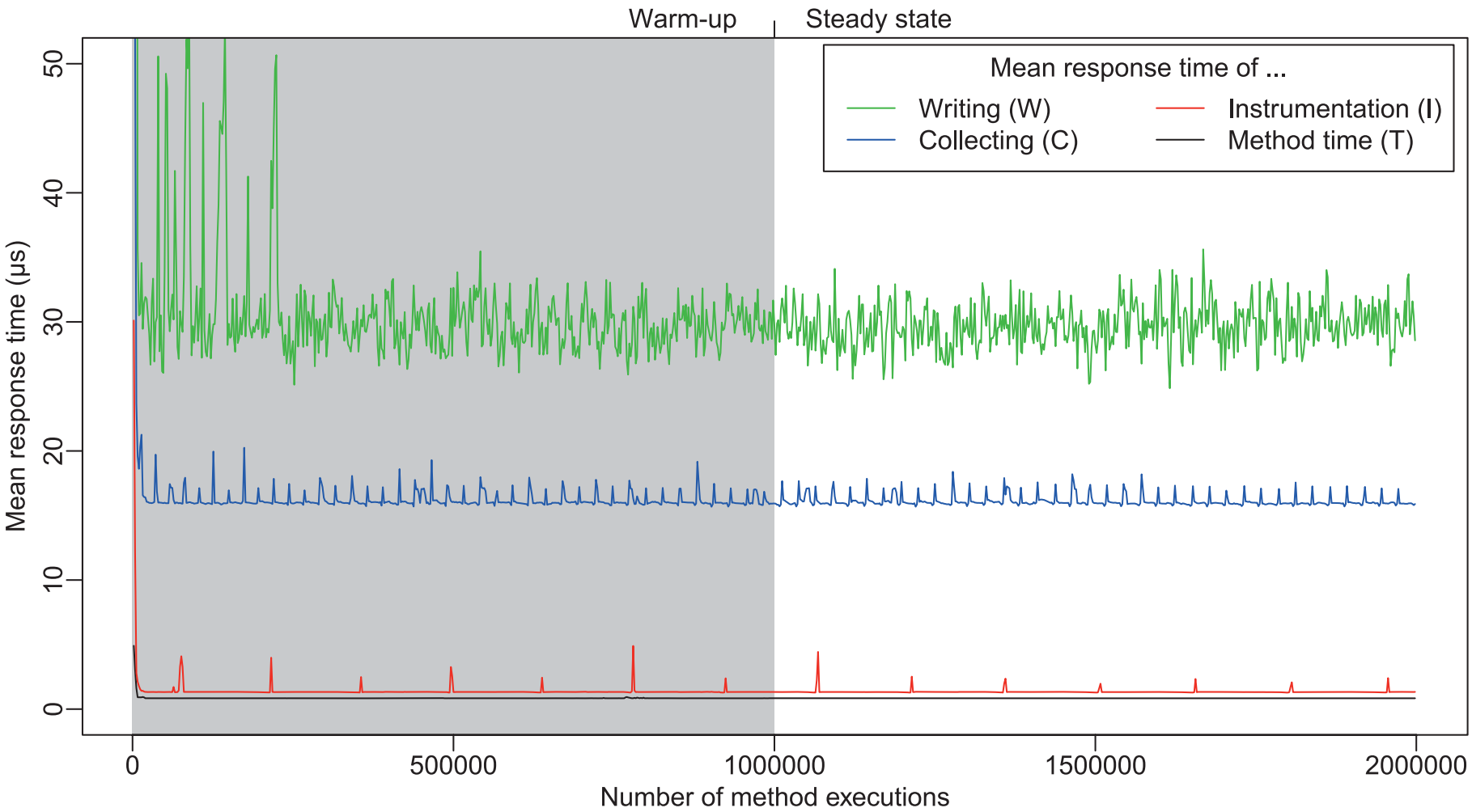

Figure 4: Time series diagram of measured timings

Overhead Reduction Tunings

- Four performance tunings (PT1 to PT4)
- Used the benchmark for structured performance optimizations
- **Goal:** Low monitoring overhead and high throughput
- Every tuning is evaluated by the benchmark
- We will see whether usable in Kieker or not

Experimental Setup

- Modifying Kieker 1.8
- X6270 Blade Server with
  - 2x Intel Xeon 2.53 GHz E5540 Quadcore processors,
  - 24 GiB RAM, and
  - Solaris 10

C A U
Christian-Albrechts-Universität zu Kiel
Technische Fakultät

	No instr.	Deactiv.	Collecting	Writing
Mean	1 176.5k	757.6k	**63.2k**	**16.6k**
95% CI	± 25.9k	± 5.5k	± 0.1k	± 0.02k
Q_1	1 189.2k	756.6k	63.0k	16.2k
Median	1 191.2k	765.9k	63.6k	16.8k
Q_3	1 194.6k	769.8k	63.9k	17.2k

Table 1: Throughput for basis (traces per second)

C A U
Christian-Albrechts-Universität zu Kiel
Technische Fakultät

- High monitoring overhead in:
  - Collection of data and
  - actually writing the gathered data
- Expensive Reflection API calls
- Reuse of signature of operations

C A U
Christian-Albrechts-Universität zu Kiel
Technische Fakultät

	No instr.	Deactiv.	Collecting	Writing
Mean	1 176.5k	757.6k	63.2k	16.6k

Table 2: Throughput for basis (traces per second)

	No instr.	Deactiv.	Collecting	Writing
Mean	1 190.5k	746.3k	**78.2k**	**31.6k**
95% CI	± 4.1k	± 4.1k	± 0.1k	± 0.1k

Table 3: Throughput for PT1 (traces per second)

- Will be used in Kieker since not impacting interfaces

C A U
Christian-Albrechts-Universität zu Kiel
Technische Fakultät

- From PT1: Queue is saturated and the monitoring thread waits for a free space in the queue
- Target: Decrease the synchronization impact of writing data
- Optimize the communication between monitoring and writer thread
- **Disruptor** instead of Java's **ArrayBlockingQueue**

	No instr.	Deactiv.	Collecting	Writing
Mean	1 190.5k	746.3k	78.2k	31.6k

Table 4: Throughput for PT1 (traces per second)

	No instr.	Deactiv.	Collecting	Writing
Mean	1 190.5k	757.6k	78.2k	**56.0k**
95% CI	± 3.6k	± 6.2k	± 0.1k	± 0.2k

Table 5: Throughput for PT2 (traces per second)

- Will be used in Kieker since only impacting communication between MonitoringController and Writers

C A U
Christian-Albrechts-Universität zu Kiel
Technische Fakultät

- From PT2: Monitoring thread is waiting for the writer thread to finish
- Target: Decrease the writing time
- Reduce the conducted work of the writer thread
- Flat record model (**ByteBuffer**s)

C A U
Christian-Albrechts-Universität zu Kiel
Technische Fakultät

	No instr.	Deactiv.	Collecting	Writing
Mean	1 190.5k	757.6k	78.2k	56.0k

Table 6: Throughput for PT2 (traces per second)

	No instr.	Deactiv.	Collecting	Writing
Mean	1 176.5k	729.9k	**115.7k**	**113.2k**
95% CI	± 2.1k	± 4.4k	± 0.2k	± 0.5k

Table 7: Throughput for PT3 (traces per second)

- Will not be used in Kieker since monitoring records now writing bytes directly to buffers (less maintainable)

C A U
Christian-Albrechts-Universität zu Kiel
Technische Fakultät

- From PT3: About 80% spent time in collecting phase
- Target: Decrease the collecting time
- Remove interface definitions, configurability, and consistence checks
- Five hard coded types of MonitoringRecords

C A U
Christian-Albrechts-Universität zu Kiel
Technische Fakultät

	No instr.	Deactiv.	Collecting	Writing
Mean	1 176.5k	729.9k	115.7k	113.2k

Table 8: Throughput for PT3 (traces per second)

	No instr.	Deactiv.	Collecting	Writing
Mean	1 190.5k	763.3k	**145.1k**	**141.2k**
95% CI	± 2.0k	± 4.0k	± 0.2k	± 0.3k

Table 9: Throughput for PT4 (traces per second)

- Will not be used in Kieker since breaking the framework idea

- At least one core was available for the monitoring
- Common threats of micro-benchmarks (relevance and systematic errors)
- Different memory layouts of programs or JIT compilation paths

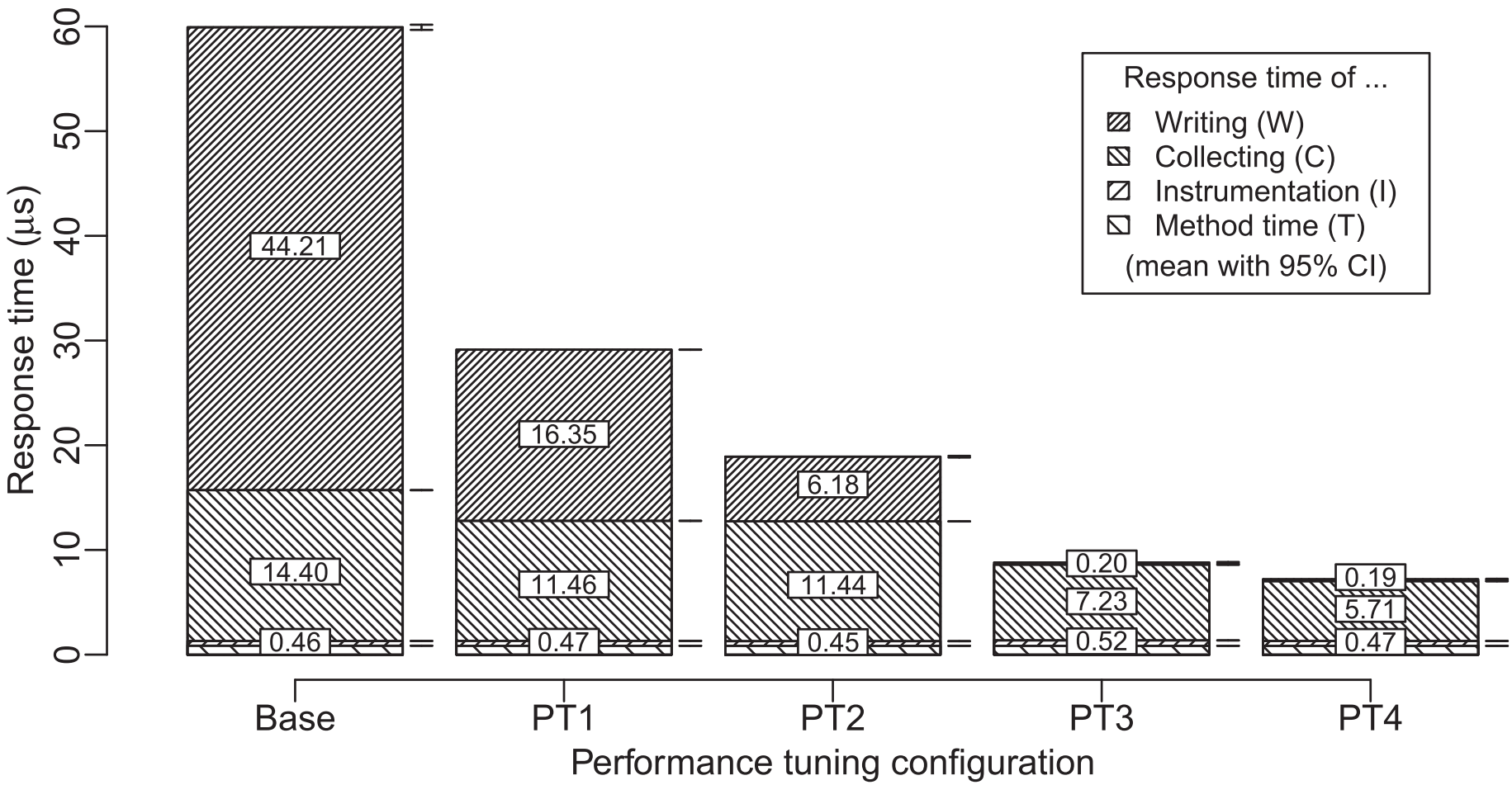

Figure 5: Overview of the tuning results in response time

- Dapper
- Magpie
- X-Trace
- SPASS-meter

Future Work

- Reduce the impact of deactivated probes by, for instance, DiSL
- Generator handling the monitoring record byte serialization
- Multi-threaded versions of our monitoring benchmark
- Compare to other benchmarks

- Proposed micro-benchmark for monitoring frameworks

- Tunings show an upper limit for the monitoring overhead
- Useful for live trace processing in the context of ExplorViz[1]

[1] http://www.explorviz.net

Jan Waller and Wilhelm Hasselbring.
A benchmark engineering methodology to measure the overhead of application-level monitoring.
In Proceedings of the Syposium on Software Performance: Joint Kieker/Palladio Days (KPDays), pages 59–68, 2013.