Refactoring Kieker's Monitoring Component to Further Reduce the Runtime Overhead

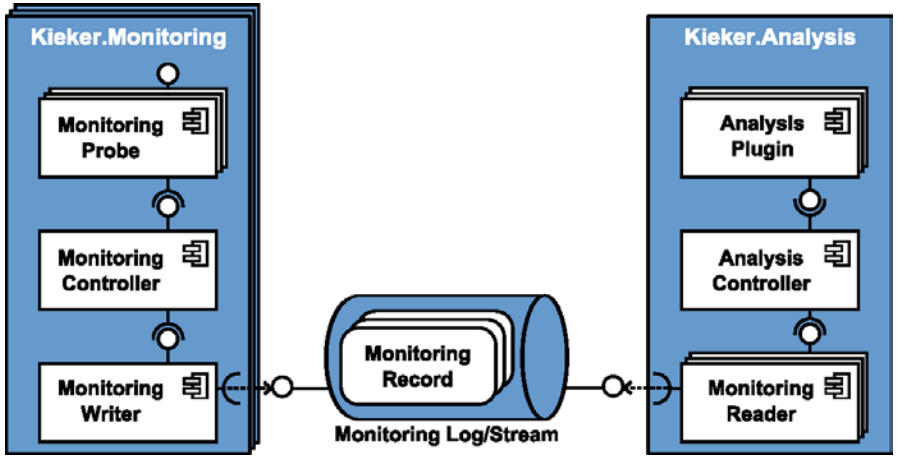
Symposium on Software Performance 2016

Hannes Strubel and Christian Wulf

08.11.2016

Software Engineering Group
Kiel University, Germany

Kieker Architecture

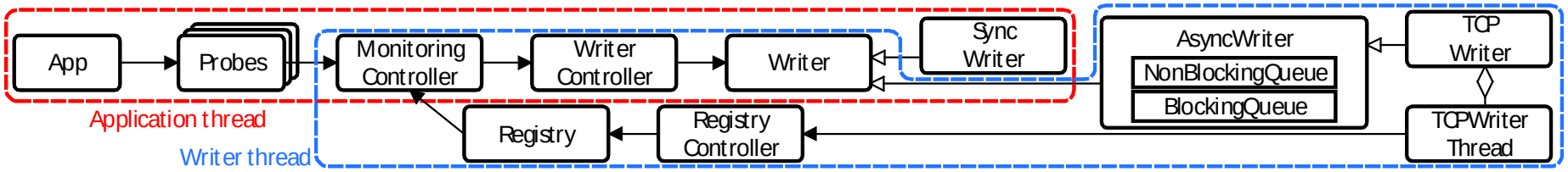


- Low monitoring overhead [WallerSSP13] MooBench
- Fast Pipe-and-Filter-based analyses (migration completed soon) [WulfSSP14] TeeTime
- Resolved performance anti-patterns [WulfSSP15]

Agenda

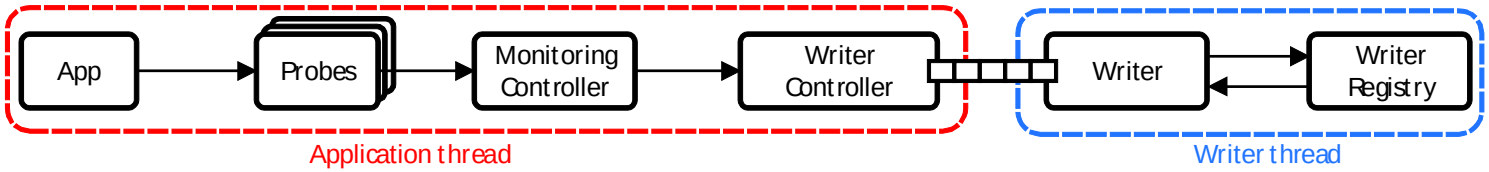
- Introduction
- Current Monitoring Component
- Refactored Monitoring Component
- Evaluation
  - Complexity
  - Runtime Overhead
- Conclusions

Current Monitoring Component



- The Application threads atomically increase the number of transferred records
- The WriterController checks for the type of the incoming record
- The used implementation of the synchronized queues is very slow
- The string registry
  - is synchronized since it is used by two writer threads
  - is maintained by the registry controller
- Each writer has to declare and to manage its own set of worker threads

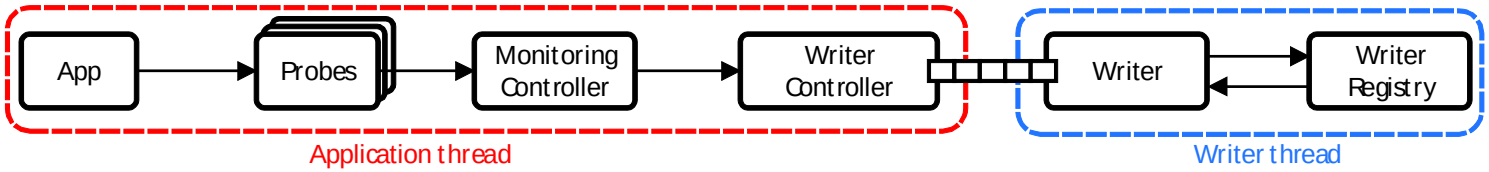
Refactored Monitoring Component



- Introduction of a new event-based system
  => replaces the atomic counter
- Removal of registry records
  => application threads do not check for record types anymore
- Avoidance of one of the two queues
- Introduction of a high-performance, lock-free queue based on JCTools[1]
  => replaces the slow blocking queue
- No synchronous writer anymore

1: https://github.com/JCTools/JCTools

Refactored Monitoring Component



- Remaining writer thread first checks a record's the string attributes
- String registry declaration on demand by each writer
- Faster, unsychronized string registry
- No need for the registry controller anymore
- Writer controller now handles the thread management

Evaluation

- Complexity
  - Using the Hypergraph-based Software Evaluation-Plugin for Eclipse[1]
- Runtime Overhead
  - Using MooBench[2]

[1] https://build.se.informatik.uni-kiel.de/eus/se/snapshot
[2] https://build.se.informatik.uni-kiel.de/kieker/moobench

Complexity Evaluation

Kieker Version	Lines of Code	Cyclomatic Complexity	Information Complexity
Current	1092	2.48	242
Refactored	(60%) 654	(70%) 1.73	(73%) 176

Runtime Overhead Evaluation

Kieker Version	No Instr.	Deact. Probe	Data Coll.	Discard Writer	TCP Writer
Current	0.087	0.478	2.607	18.932	19.715
95%-ci (±)	0.000	0.005	0.007	0.014	0.015
Refactored*	0.088	0.468	2.335	12.424	16.855
95%-ci (±)	0.000	0.004	0.007	0.007	0.035
Refactored	0.083	0.472	2.384	2.785	3.265
95%-ci (±)	0.000	0.006	0.012	0.009	0.011

Conclusions

- Refactored architecture of the monitoring component
- Information complexity reduced to 73 %
- Runtime overhead reduced to 17 % (!)

http://kieker-monitoring.net

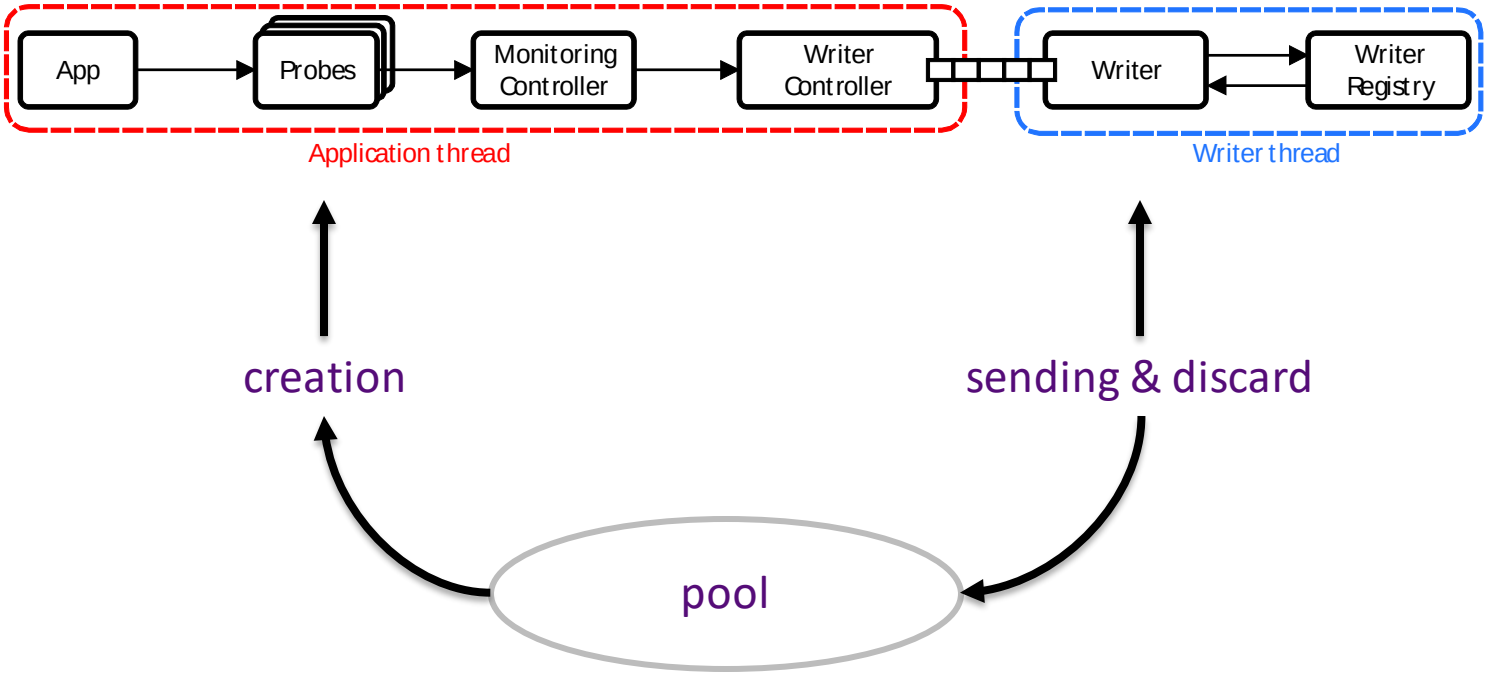
http://teetime.sourceforge.net

Future work

- Record pooling to reduce pressure on GC
- Record data pre-filling to minimize data collection at runtime

Record Pooling

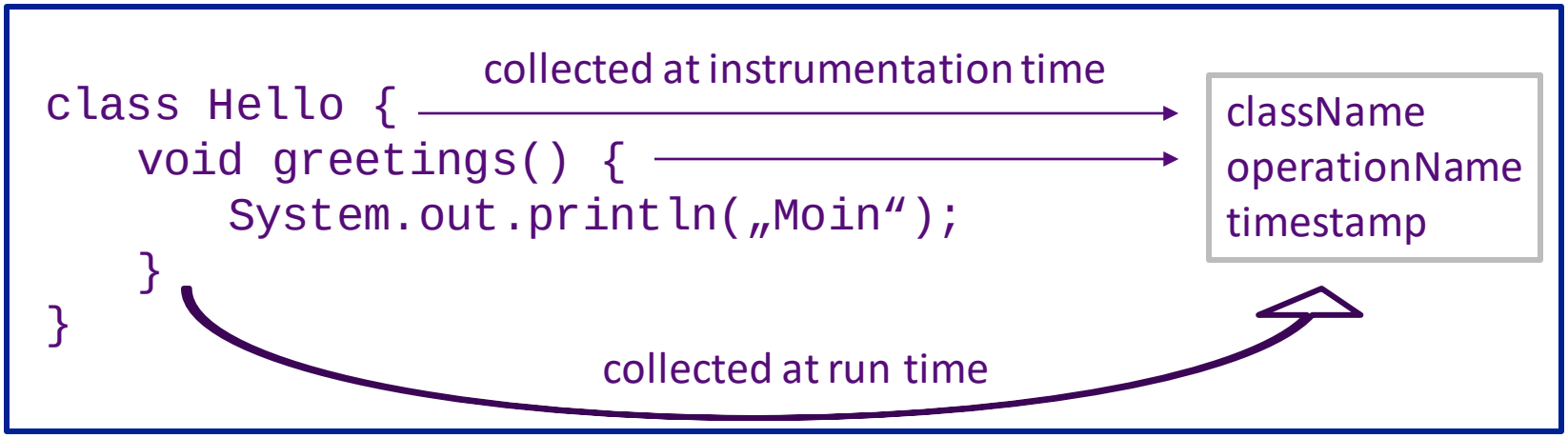
Lifetime of a record within the monitoring component



Record Data Pre-Filling

Aspectj limited: unable to intercept weaving process

=> No access to static code information at weaving, such as class and method name



Proposition: instrumentation via a low-level framework such as ASM/Javassist

References

[WallerSSP13] Waller, J. und Hasselbring, W., „A Benchmark Engineering Methodology to Measure the Overhead of Application-Level Monitoring", In: Proceedings of the Symposium on Software Performance: Joint Kieker/Palladio Days (KPDAYS 2013), 2013, Karlsruhe, Germany.

[WulfSSP14] Wulf, C., Ehmke, N. C. und Hasselbring, W., „Toward a Generic and Concurrency-Aware Pipes & Filters Framework, In: Proceedings of the Symposium on Software Performance: Joint Descartes/Kieker/Palladio Days, 2014, Stuttgart, Germany.

[WulfSSP15] Wulf, C. und Hasselbring, W., „Software Performance Anti-Patterns Observed and Resolved in Kieker", In: Proceedings of the Symposium on Software Performance: Joint Developer and Community Meeting of Descartes/Kieker/Palladio, 2015, Munich, Germany.