Performance Evaluation of Aspect-Oriented Programming Weavers

Michel S. Soares[1](✉), Marcelo A. Maia[2], and Rodrigo F.G. Silva[2]

[1] Computing Department, Federal University of Sergipe, São Cristóvão, Brazil
mics.soares@gmail.com
[2] Computing Faculty, Federal University of Uberlândia, Uberlândia, Brazil

Abstract. Aspect-oriented programming (AOP) was proposed with the purpose of improving software modularization by treating crosscutting concerns. Since its introduction, there is no consensus about the impact on performance of the use of AOP techniques to deal with crosscutting concerns. This article explores further the evaluation of performance by proposing a systematic literature review to find out how performance is affected by the introduction of aspects. Then, an experiment is performed to find results about the performance of AOP and weavers. This experiment showed the assessment of several versions of an application. According to this study, the difference concerning resource consumption through variation of weavers can be considered irrelevant considering web applications.

Keywords: Aspect-oriented programming · Systematic literature review · Crosscuting concerns · Performance · MVC framework

1 Introduction

Aspect Oriented Programming (AOP) [1] was proposed as an attempt to deal with crosscutting concerns aiming to improve modularization. AOP has gained importance since its introduction to implement crosscutting concerns, with varying degree of success [2–5]. Within AOP, crosscutting concerns are implemented as aspects and are further weaved into code. The way aspects are weaved into code may affect performance as the weaving process introduces new code to the original programs.

The impact on performance, caused by AOP techniques, has motivated previous works in scientific literature. Liu [6] showed that the aspect-oriented approach does not have significant effect on performance, and that in some cases, aspect-oriented software even outperform the non-aspect one. Additionally, introduction of a large number of join points does not have significant effect on performance. Remko [7] assessed the performance effects between programs created by a weaver and a hand-coded version. This work came to the conclusion that simple advices give no real performance penalties, but the more sophisticated advices are, the slower they become. Kirsten [8] compared the four leading

AOP tools at the time (2004) and, when it comes to performance, he postulated that, in general, code with aspects performs similarly to that of a purely object-oriented solution, where the crosscutting code is scattered throughout the system.

The use of AOP to implement crosscutting concerns and its impact on performance is the motivation for this study. First, a systematic literature review is performed. The goal of this systematic review is to understand the extent of the impact of AOP on the performance of software systems, if there is an impact. The purpose of the experiment described following the review is to find results about performance and aspect-oriented implementations. Considering the fact that AOP techniques add additional complexity in software in order to treat cross-cutting concerns, this addition can generate an overhead in software execution, leading to impact on performance.

Considered factors or independent variables for evaluating performance in AOP software include the weaver, the type of weaving, the type of advice, the number of join points, the size measured in lines of code, and the number of classes to be loaded in the load-time weaving process. In order to evaluate the impact of AOP techniques on performance, this article proposes the evaluation of a major factor that may impact these techniques: the type of weaving. The factors *number of lines of code*, *types of advices*, *number of joinpoints* and *number of classes* to be loaded during load-time weaving will be considered in future research. Therefore, the hypotheses of this article is that changing the weaving process sensitize the outcomes.

2 Protocol for Systematic Review

The main question that motivated the systematic literature review [9] we conducted in this paper is: *Does the use of aspect-oriented techniques to implement crosscutting concerns impact software performance ?* A derived research question is "*If the impact exists, how meaningful is it ?*". The answer to both questions could help developers to reason about the feasibility of the use of AOP techniques to handle crosscutting concerns on architectures where performance is itself a concern. The systematic review started by searching in a number of software engineering conferences and journals. The search was performed considering publications in the past 6 years. The chosen conferences were: AOSD (International Conference on Aspect-Oriented Software Development) and ICSE (International Conference on Software Engineering). The chosen journals were: JSS (Journal of Systems and Software), IST (Information and Software Technology), SCP (Science of Computer Programming), TSE IEEE (IEEE Transactions on Software Engineering), TOSEM (ACM Transactions on Software Engineering Methodology). ENTCS (Electronic Notes in Theoretical Computer Science), which can be considered a series, was also included.

The search string was ("Aspect-oriented programming" AND "performance"). The search has retrieved 338 papers. From these 338 papers, a subselection has been made with the purpose of separating those relating AOP with

Table 1. Search results and selected papers.

Publication	Retrieved papers	Relevant papers	Selected papers	Data source
JSS	38	2	1	ScienceDirect
IST	32	10	5	ScienceDirect
SCP	32	2	1	ScienceDirect
TSE	1	1	1	IEEExplorer
TOSEM	21	3	1	ACM Digital Library
ENTCS	26	3	1	ScienceDirect
AOSD	127	9	3	ACM Digital Library
ICSE	61	2	2	ACM Digital Library
Total	338	32	15	

Table 2. Selected papers.

	Venue/year	Reference
1	ICSE/07	[10]
2	SCP/08	[11]
3	IST/09	[12]
4	ENTCS/09	[13]
5	IST/09	[14]
6	ICSE/09	[15]
7	AOSD/09	[16]
8	JSS/10	[17]
9	ACM/10	[18]
10	IST/10	[19]
11	AOSD/10	[20]
12	IST/10	[21]
13	AOSD/10	[22]
14	IEEE/12	[4]
15	IST/12	[23]

any performance metrics. In a first step, 32 papers were selected and classified as relevant. For the first selection, the title, the keywords and the abstract were read. If the subject was pertinent to AOP and performance, the introduction and the conclusion were read as well. In case of doubt about the relevance of the paper, specific keywords were searched in the paper, such as *aspect*, *crosscutting* and *performance*. There were also relevant papers which used other terms, including *cost*, *payload* and *overhead* when considering assessment of performance of some AOP technique, and in those cases, they were selected too.

In a second step, of the 32 relevant papers, only those ones which assess the performance of implementation of some crosscutting concern were selected to be fully read. As a result, 15 papers were selected in total. Several types of concerns have been classified by the papers as crosscutting concerns, even though some of them were domain specific. However, papers which had crosscutting concerns implemented through some AOP technique but which did not consider any assessment of the used technique(s), or this assessment was incomplete, were discarded. The summary of the filtering process can be seen in Table 1 and the final selection of papers is presented in Table 2.

3 Systematic Review Results

All 15 selected papers were fully read for the evaluation. The selected papers were evaluated based on two sets of criteria: Application Type and Performance.

3.1 Application Type Criteria

The first set of criteria concerns about Application Type and encompass the following metrics: number of assessed studies, lines of code (size, in LOCs), original programming language (Original PL), aspect programming language (Aspect PL) and application domain.

The application type is related to the type of application of the case studies or experiments which have been assessed by the papers. The following types were retrieved from the papers: Middleware, Web Service, Embedded, Platform, System or Application, Language or Extension (Language) and Framework. Cases where their case studies were described as Monitoring Systems were classified as System or Application. Papers which did not mention the application type of their experiments have been classified under the closest definition of these ones already mentioned. The number of assessed studies indicates only those studies that were implemented by some AOP technique and were assessed by some kind of metric.

The application domain includes: e-commerce, industrial application, Office, Bank and Generic. Cases where there is no specific domain, for example a toolkit or a language extension, were classified as Generic. Some papers, mostly in application type, did not mention the application domain and were also classified by proximity.

The summary of studies is presented in Table 3. Cases where no metric was presented or in which it was not possible to identify were classified as not available (NA).

3.2 Performance Criteria

The second set of criteria concerns *Performance*. Four metrics were extracted from the papers: weaving type, implemented crosscutting concerns, used performance method, and performance overhead.

Table 3. Summary of studies.

Type	Article	Studies	Size	Original PL	Aspect PL	Domain
Middleware	[17]	2	NA	Java	AspectJ	Generic
	[15]	3	12.7KLOC, 113Kb, 190Kb		FlexSync	Industrial
Embedded	[13]	1	NA		ObjectTeams, Java	
	[23]	2	NA	GPL	NA	
System or Application	[21]	1	NA	Java	JBoss AOP	Generic
	[14]				AspectJ	Office
	[16]	1	46KLOC		Java, AspectJ	Generic
	[11]	1	NA		KALA	Bank
	[10]				AspectJ, JBoss AOP	Industrial
	[4]	3	1.6KLOC, 13.9KLOC, 51.6KLOC	C++	AspectC++	
Language	[20]	1	118KLOC	JavaScript	AspectScript	Generic
	[18]	2	NA	Java	NA	Industrial
Framework	[22]	1			Compatible with AspectJ	Generic
Platform	[12]			NA	NA	E-commerce
Web Service	[19]			Java	AspectJ	

The weaving type indicates the type of weaving performed by the studies in the papers. Two main kinds were considered: *Compile-time* and *Runtime weaving*.

Several kinds of crosscutting concerns were retrieved from the papers. There were cases where the crosscutting concerns were domain specific. Papers which treated only one concern in the study prevailed, but there were cases where more than one concern was considered, one of them domain specific [4].

The performance methods retrieved were the measurements of running or execution time (ext), business operations per second (bos), average memory overhead (avmo), CPU usage (cpu), qualitative observation of the overall execution (obs), parsing time (pat), and average number of method calls per second (met). Some papers presented more than one assessed variable. In these cases, when there was performance reduction, the considered measurements were based on the worst case. Some cases related the performance overhead as negligible (negl).

The results of the performance assessment performed by the papers are presented in Table 4. In the Performance Overhead column, the "+" and the "−" signs means decrease and increase in performance, respectively. If a sign is followed by negl, it means that the paper reported a degradation or gain in performance, but this result is negligible according to the authors.

Table 4. Performance analysis of studies.

Weaving	Article	Crosscutting concerns	Performance Method	Performance overhead
Run-time	[17]	Stylistic	ext, avmo	ext: -negl, avmo: +1.03x to 1.1x
	[15]	Synchronization	bos	negl
	[13]	Maintainability, Extensibility and Reusability	ext	-2x for 100 instances
	[21]	Reconfigurability	ext, cpu, avmo	ext: +1.1x to 1.22x, cpu: +NA, avmo: +NA
	[14]	Monitoring	Qualitative Observation	not observable
	[16]	Security	pat	+up to 1.16x
	[20]	Expressiveness	met, cpu	met: +up to 16.1x, cpu: negl
	[11]	Transaction Management	NA	+ NA
Compile-time	[4]	Caching, CheckFwArgs, Excepter, Singleton, Tracing, CadTrace, FwErrs, FetTypeChkr, Timer, UnitCvrt, ViewCache, ErcTracing, QueryConfig, QueryPolicy	ext, avmo	ext: + up to 1.18x, avmo: + up to 1.15x
	[22]	Comunication between threads	ext	+ factor up to 31.08x
	[19]	Device adaptation	ext	negl
Compile-time / runtime	[10]	Constraint validation	ext	+ varies according to approach
Domain Specific	[18]	Cache	met	+1.015x
NA	[23]	Safety	NA	+ NA
	[12]	Security	ext	+ varies according to approach

3.3 On the Target Applications

From the first set of criteria, related to *Application type*, it is possible to conclude that most papers, 10 out of 15, assessed only one study or experiment. However, only four of them showed the LOC or size of their assessed studies. The two studies that have evaluated more systems, evaluated three small-scale systems (at most 50KLOC or 190 Kb). The larger evaluated system had 118KLOC. One hypothesis for such lack of large scale studies is that AOP is not extensively adopted such as OOP or procedural programming. Therefore, the low adoption from the community restricts the availability of large systems for experimentation.

The prevailing application type was *System* or *Application*. That is reasonable to expect because in general this kind of applications are more frequent and more accessible. The prevailing application domain was *Industrial applications* followed by applications with no specific domain, hereby classified as Generic. We can observe that there is reasonable variability in terms of *Application Type* and *Application domain*. We could observe that in Middleware software the overhead was negligible. In the category *System* or *Application*, there is a tendency of more impact in the performance.

The LOC seems not to influence because the larger studies systems had negligible impact on execution time and CPU performance. The Application Domain also seems not to influence the performance because of the high variation in the results. The application domain did not present a clear influence in performance. The *Industrial* domain, which has the larger number of studies, had also presented negligible and positive impact in performance.

Finally, concerning the implemented crosscutting concerns in the applications, there was no prevailing concern in the studies, and surprisingly, none of the studies implemented common concerns such as Logging or Exception Handling. This can be an indicative that the studied cases were not representative in terms of typical aspect-oriented software.

3.4 On the Used Programming Languages

The prevailing original programming language was Java with 11 out of 15 studies and the prevailing aspect programming language was AspectJ with 6 out of 15 studies. JBoss AOP was present only in two cases, and Spring AOP was not used in any of them. Considering the impact of the programming language in performance, we can observe that the original programming language that has significant number of studies is Java, but there was no clear indication that Java is an influence factor. In the same way, AspectJ, which is the prevailing aspect language, has shown no direct influence on the performance because it presented either negligible or positive impact in performance. Although only two studies were carried out with JBoss AOP, both studies have shown a positive impact in performance.

3.5 On the Type of Weaving

From the second set of criteria, it is possible to conclude that run-time is the most common weaving type process presented by papers in the experiments. One of the reasons for this choice, instead of compile-time weaving, is the fact that runtime weaving allows aspects to be added to the base program dynamically, which is better for a context-aware adaptation of the applications [13]. Some papers not only used the runtime weaving process but also extended it to adequate the process to their studies. Also concerning the weaving process, the papers in general postulated that runtime weaving requires more effort at runtime, impacting on performance, but no proofs about this assumption were found in this research.

3.6 On the Experimental Setting of Reviewed Papers

Papers assessed their experiments in different ways, but the prevailing performance metric was measurement of execution time (ext). The performance overhead varied according to a set of variables such as the used approach, implemented concern, the aspect programming language, weaving type process and the used aspect weaver.

The workload is strongly influenced by the target application, which defines several other sub-factors: the kind of join points, pointcuts and advices, the ratio of occurrence of AOP constructs and the other non-AOP constructs, the requirement of the application for specific type of weaving. Considering the space for combination of levels for these factors, it is challenging (if not impossible) to find a real world application (or a set of them) that can have all the possible levels. Therefore, an alternative could be the design of a synthetic application that could be use as a benchmark for performance evaluation of AOP techniques. This benchmark would need to be meaningful to mimic real world scenarios and would need to be comprehensive to guarantee that all important factors and their respective levels would be considered.

4 Experimental Settings

In order to produce further evidences on the impact of the type of weavers in the performance of AOP programs, we decided to conduct our own experiment. This section explains the conditions of the experiment and how the experiment was executed. The section is organized as follows. In Subsect. 4.1 the environment in which the experiment was performed is explained. Subsection 4.2 shows the versions of the case study. Subsection 4.3 explains about the scenario of the experiment. Subsection 4.4 shows the plan of the experiment designed for executing the tests.

4.1 Environment

The original software in which the experiment was performed is a web application system called SIGE. Designed in 2011, it has the purpose of automating the management of information of academic and administrative units at Federal University of Uberlândia. The System was developed in Java. SIGE uses two frameworks, Struts (version 2.3.3) and Spring (version 3.0.0). SIGE also access a DB2 database.

The crosscutting concern implemented by AOP is Logging, responsible for logging some methods in service and DAO layers of the system for purposes of auditing. In order to perform the experiment, a copy of the last version of SIGE was produced and named as version V0. Version V0 has as size 297.915 LOC, considering files of the following extensions: java, jsp, js, html, xml and css. From version V0, other versions were built. This work adopted the same idea of inheritance from Object Orientation for extended versions. Each extended versions preserve the same properties of the parents and override the interested properties so the proposed factors could be measured and analyzed.

Three weaving processes were considered regarding AOP. These are compile-time weaving, load-time weaving, and runtime weaving. The class responsible for implementing the Logging aspect is the AspectProfiler class. The adopted style for enabling AOP into the class was the AspectJ annotation Style, provided by the Spring AOP AspectJ support. Each method is composed of an advice annotation where a *pointcut* expression is used to match the join points, which is a method execution for all expressions. Despite the fact that AspectJ has several definitions for join points, those used in the AspectProfiler class always represent a method execution because of the Spring AOP framework restriction which limits join points to be only method executions. This kind of join point was adopted as default in order to enable comparisons among the different considered weavers. This class was the only class used to implement the Logging aspect and had its content varied in versions, according to the purpose of each version. Each phase of the experiment explains how this content was modified for its purposes.

Two types of invocations were considered to perform the experiment, internal and external invocations. External invocations are generated by the JMeter[1] tool while internal invocations are generated by a JUnit class, which are explained in details in Sect. 4.4.

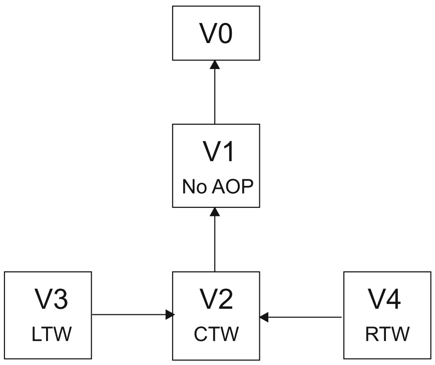

Fig. 1. Versions for the case study.

Concerning system settings, all tests were executed under the same hardware and software conditions. The hardware was composed of one notebook with Intel Core i5-2410M 2.30 GHz processor and 4 GB of RAM memory running Ubuntu 12.04 LTS operational system. The used Java version was 1.7.0. Before all test measurements, memory and swap conditions were verified through Ubuntu System Monitor and were always under 55 % and 2 % respectively. The running processes during applications were, besides regular Ubuntu processes, Eclipse, System Monitor, LibreOffice calc, VisualVM and, in specific stages, JMeter.

[1] https://jmeter.apache.org/.

Despite the fact that only 4 assessed versions of the experiment access the database, a running DB2 database was necessary for the application context to run. The version of the running DB2 database was 9.7. Network was disabled during tests in order to avoid CPU interruptions.

4.2 Versions

Figure 1 shows the design of proposed versions. The image is divided into versions that will be compared to each other. It contains the original version (version V0), one version with no AOP involved (version V1) and three other versions weaved by three different weaving processes, which are compile-time weaving (CTW), load-time weaving (LTW) and runtime weaving (RTW), respectively versions V2, V3 and V4.

4.3 Scenario

The chosen scenario for the experiment was the user authentication scenario. This scenario was chosen because it is composed of several method calls in different system layers and it does not present complex business rules.

The scenario was refactored so that the invocations do not access the database. The reason for not using the database access is due to the fact that the resources consumed by I/O operations would influence the measurements of the logging aspect. Likewise, at the data access object (DAO) layer, all methods of the scenario returns with the minimum of processing and simulate a successful authentication. This refactoring did not alter the purpose of the methods or the scenario sequence of operations. The resulting version, named V1, was used to build all other versions where factors were varied so the outcomes could be measured.

4.4 Plan of the Experiment

In order to measure the outcomes, a default experiment plan was made. This plan is composed by a parameter with fixed value and parameters with variable values, which are set in each system version. The parameter with fixed value is the number of test executions, which defines how many times the test is executed to generate outcomes data and was set to 15.

One of the parameters with variable value is the *loopCount* variable, present in the JUnit test class for internal invocations and configured in the JMeter experiment plan for external invocations. The loop count defines how many times each internal invocations or HTTP request is executed on the scenario in a test. It was initially set to 10.000. The criteria to choose is because this number should be high enough to emphasize the small impact on performance caused by some interested factor, but low enough to allow the assessment of the set of invocations in an acceptable time. From a set of considered options, which were 1, 10, 100, 1.000, 10.000 and 100.000, the number 10.000 was the best choice for versions with disabled database and which used internal invocations as default measuring method.

CPU, memory and time were measured in two forms, depending on the type of invocation. For internal invocations, the execution of each test is launched through Eclipse. A breakpoint is set at the beginning of test execution in JUnit test class, where the start method of StopWatch is invoked and starts time counting. This breakpoint is necessary for the process inspection through VisualVM tool. When the JUnit running thread stops at the breakpoint, VisualVM is configured to inspect only memory and CPU in the application process. After these steps, breakpoint is released, the process run and the current aspect mechanism perform over the scenario method executions. After the end of the process execution, StopWatch class calculate the elapsed time in milliseconds and shows it at Eclipse console while VisualVM generates the real time report of memory and CPU history. CPU and memory are manually retrieved and, the highest values of the graphics, respectively, in percentage and megabytes (MB), were always considered.

For external invocations, no breakpoint is necessary for VisualVM to inspect CPU and memory. After the launch of tomcat through Eclipse debug, VisualVM is capable of inspecting the running process of tomcat before the beginning of the invocations. After configuring VisualVM to inspect CPU and memory on tomcat process, the JMeter experiment plan is invoked. After the execution of the test, JMeter report provides the time report while CPU and memory are retrieved manually, as done for the internal invocations.

5 Design of the Experiment

The experiment plan was run in each phase with the proper parameters. This Section explains the design of the experiment, showing the phases, their details and purposes.

5.1 Versions of the Software

The purpose of this phase is the generation of versions containing different weaving processes and serves as template to be used for building the other versions. In this Section the building process of versions V1, V2, V3 and V4 is presented.

Version V1. The purpose of this version is to establish baselines of values related to the dependent variables to be compared with values obtained in other phases of the experiment.

Version V2. From version V1, version V2 was built. This version has the purpose of assessing the impact on performance by adopting the **compile-time weaving process**, provided by the AspectJ weaver. The results of this phase are compared to the results of version V1, this way it is possible to assess if the weaver AspectJ sensitizes the dependent variables through the compile-time weaving process. In this version all log generation messages were removed from the scenario methods. The weaver was provided by the plugin of the AspectJ Development Eclipse Tools. This plugin, when enabled, forces the project to re-compile, thus injecting the aspect code bytecodes to the classes at compile time.

The following parameters were set:

– Weaver: AspectJ compile-time weaver;
– Number of advices: 6;
– Type of advices: 2 before, 2 around, 2 after, having log messages invoked only on before advices;
– PointCut Expressions: interception of all methods present in service and DAO packages.

Version V3. From version V2, version V3 was built. This version has the purpose of assessing the impact on performance by adopting the **load-time weaving process**, provided by the AspectJ weaver. Like in version V2, the results of this phase are compared to the results of version V1 in order to verify if this weaver, using this weaving process, sensitizes the dependent variables. In this version, the AspectJ plugin, which is responsible for compiling the aspects, was disabled and the weaving type paremeter was overridden to the load-time weaving process. The load-time weaving process is used in the context of Spring Framework. AspectJ is responsible for weaving aspects into the application's class files as they are being loaded into JVM. From an aop.xml file created in META-INF folder, which was added to the build path of the project, with the necessary configuration to weave aspects of AspectProfiler class into the project classes, this file specifies the loading of two classes to the JVM, and one DAO class containing the methods of the scenario.

Version V4. From version V2, version V4 was built. This version has the purpose of assess the impact on performance by adopting the **runtime weaving process**, provided by the Spring AOP framework. Like in versions V3 and V2, the results of this phase are compared to the results of version V1 in order to verify if this weaver, using this weaving process, sensitizes the dependent variables. In this version, the AspectJ capability provided by the AspectJ plugin was disabled and the weaver overridden to Spring AOP.

This phase has the purpose of analyzing AOP together with two layers which impacts performance significantly, which are the chain of mechanisms present in the front end controller provided by the MVC framework, in this case Struts 2, and the access to the database, here represented by DB2.

Previous tests reported that the time spent to assess each external invocation of this version varied between 6000 and 7000 ms. The time consumed to assess the initial value for the loop count parameter, which is 10.000, would be impracticable, which addresses the loop count value to be overridden to a new value, for less. Besides this difficulty on measuring time, the maximum values of CPU and memory are manually selected on the chosen profiling tool, which is in this case VisualVM, and the higher this value, the higher the difficulty to select these maximums. However, in terms of assessing the impact on performance, the lower this value, the less the impact on performance would be noticed, which addressed this number to be 50.

6 Results

This Section presents the results of versions for each region of the map of versions for the proposed factors, as follows.

The weaver factor was varied in versions of the software. The values of the measures are shown in box plot graphs and grouped by dependent variable. In Figs. 2, 3 and 4 graphs are represented containing the consumption of time, CPU and memory for versions V1, V2, V3 and V4 respectively.

Results of versions V2, V3 and V4 were compared with results of version V1. Table 5 shows this comparison to the averages of the dependent variables.

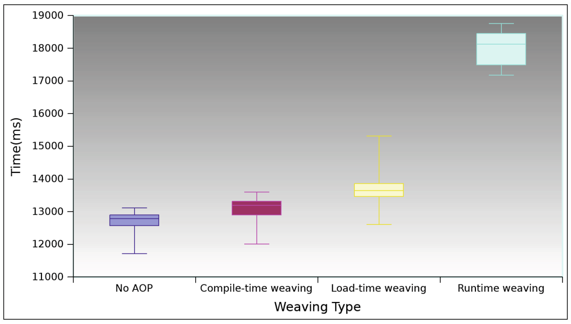

Fig. 2. Time consumption for versions V1, V2, V3 and V4.

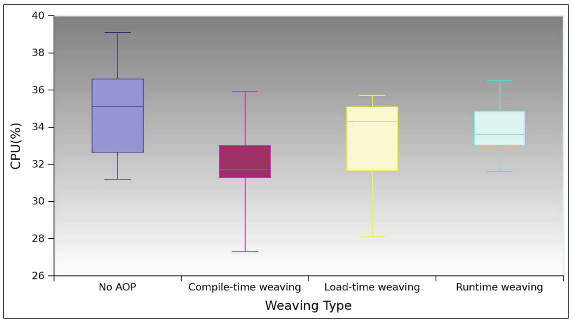

Fig. 3. CPU consumption for versions V1, V2, V3 and V4.

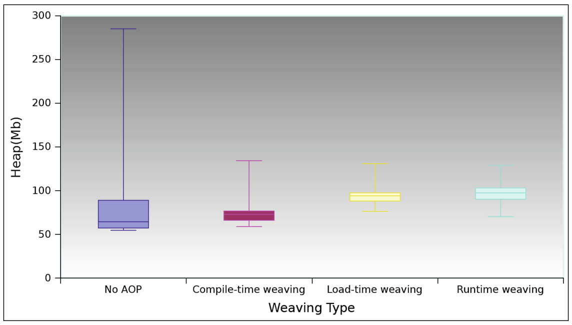

Fig. 4. Memory consumption for versions V1, V2, V3 and V4.

Table 5. Comparison between averages of dependent variables of versions V2, V3 and V4 with V1.

Compared to V1	Time(%)	CPU(%)	Memory(%)
V2	+3, 41	−8, 5	+3, 24
V3	+7, 32	−4, 56	+30, 69
V4	+41, 40	−2, 58	+39, 93

7 Conclusions

The systematic review and the further analysis of the retrieved papers presented in this article show that there are few experiments concerning AOP and performance in scientific literature. More specifically, too few experiments were reported about the performance of AOP techniques when implementing crosscutting concerns. From the results, it is clear that there is no prevailing implemented concern in the studies. On the contrary, most of the implemented concerns were domain specific. Most papers postulated that runtime weaving requires more effort at run-time, impacting on performance. In order to produce further evidences on the impact of the type of weavers in the performance of AOP programs, we decided to conduct our own experiment. The experiment showed the assessment of several versions of an application, where measurements were made with and without the presence of heavy system layers in terms of resource consumption which are the database and the whole chain of mechanisms of the MVC framework.

The variation of weavers showed that the weaver is a factor which sensitizes performance. When analyzed isolated, in other words, without the presence of the resources consuming mentioned elements, weavers showed a performance variation of about 40 % from one to another in two outcomes in certain circumstances where the pointcuts match were high.

Results of this experiment have shown that, in general, in applications which have database calls and/or MVC frameworks, the impact on performance caused by AOP solely could be considered to be negligible. However, this impact could be meaningful for applications without these layers and where AOP would be widely used. According to this study, the difference concerning resource consumption through variation of weavers can be considered irrelevant considering common web applications, but nothing can be confirmed about embedded applications where resource consumption might be limited.

The analyzed crosscutting concern was Logging. However, the focus of the analysis was on the AOP interception mechanism of each tool and not in the AOP implementation of the crosscutting concern itself. The implementation through AOP of crosscutting concerns by frameworks such as Spring could be more studied and compared to manual implementations with regard to performance. Future works could also study the performance of AOP techniques for embedded applications, where hardware conditions are limited.

Threats to validity were considered as follows. The considered outcomes were measured through three different tools. Time outcome was provided by StopWatch class on internal invocations and displayed at Eclipse console, and by JMeter on external invocations, displayed on a summarized report. CPU and memory, however, were not provided directly by any tool. Differently from time, which could be measured in absolute values, CPU and memory varies during the tests. As VisualVM did not provide the average of each one in a time shift, the used metric for obtaining these values was a manually selection of the maximum values of the graphs generated by VisualVM for each one. Measurements where the maximum values were far higher from the average values represent a threat to validity, once that these top points could not represent the real behaviour of the outcomes related to the application. Besides, mistakes on manual selection could have influenced wrong results.

The external invocations generated by JMeter were configured to be sequential. Despite the fact that on real production environments calls to the application generate multiple threads working simultaneously, sequential calls were more proper for this experiment because they do not require a possible extra effort to be scheduled. Besides, in order to achieve a possible impact on performance, the number of calls needed to be high in most cases, considering that the impact provoked by AOP instrumentation was discrete. This high number of calls would not be supported by the web container in case these calls were generated by multiple threads simultaneously. This restriction addressed the experiment to be based on sequence calls only.

Acknowledgements. The authors would like to thank CAPES (www.capes.gov.br), CNPq grant 475519/2012-4, CNPq Grant 445500/2014-0, FAPEMIG Grant APQ-01589-11, and FAPEMIG grant APQ-2086/2011 for the financial support.

References

1. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., Irwin, J.: Aspect-oriented programming. In: Akşit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)

2. Ali, M.S., Ali, M., Babar, L., Chen, K.-J., Stol, A.: A systematic review of comparative evidence of aspect-oriented programming. Inf. Softw. Technol. **52**, 871–887 (2010)
3. Przybylek, A.: Impact of aspect-oriented programming on software modularity. In: Proceedings of the 15th European Conference on Software Maintenance and Reengineering, pp. 369–372 (2011)
4. Mortensen, M., Ghosh, S., Bieman, J.: Aspect-oriented refactoring of legacy applications: an evaluation. IEEE Trans. Softw. Eng. **38**(1), 118–140 (2012)
5. França, J.M.S., Soares, M.S.: A systematic review on evaluation of aspect oriented programming using software metrics. In: ICEIS 2012 - Proceedings of the 14th International Conference on Enterprise Information Systems, vol. 2, pp. 77–83 (2012)
6. Liu, W.-L., Lung, C.-H., Ajila, S.: Impact of aspect-oriented programming on software performance: a case study of leader/followers and half-sync/half-async architectures. In: Proceedings of the 2011 IEEE 35th Annual Computer Software and Applications Conference, COMPSAC 2011, pp. 662–667 (2011)
7. Bijker, R.: Performance effects of Aspect Oriented Programming. Twente University, Technical report, Enchede, The Netherlands (2005)
8. Kirsten, M.: AOP@Work: AOP tools comparison, Part 1: Language mechanisms, Technical report, IBM Developer Works (2005). http://www-128.ibm.com/developerworks/java/library/j-aopwork1/
9. Kitchenham, B.: Procedures for Performing Systematic Reviews, Keele university. Technical report tr/se-0401, Department of Computer Science, Keele University, UK (2004)
10. Froihofer, L., Glos, G., Osrael, J., Goeschka, K.M.: Overview and evaluation of constraint validation approaches in java. In: Proceedings of the 29th International Conference on Software Engineering, ICSE 2007, pp. 313–322, Washington (2007)
11. Fabry, J., Tanter, É., D'Hondt, T.: KALA: kernel aspect language for advanced transactions. Sci. Comput. Program. **71**(3), 165–180 (2008)
12. Georg, G., Ray, I., Anastasakis, K., Bordbar, B., Toahchoodee, M., Houmb, S.H.: An aspect-oriented methodology for designing secure applications. Inf. Softw. Technol. **51**(5), 846–864 (2009)
13. Hundt, C., Glesner, S.: Optimizing aspectual execution mechanisms for embedded applications. Electron. Notes Theor. Comput. Sci. **238**(2), 35–45 (2009)
14. Ganesan, D., Keuler, T., Nishimura, Y.: Architecture compliance checking at runtime. Inf. Softw. Technol. **51**(11), 1586–1600 (2009)
15. Zhang, C.: FlexSync: an aspect-oriented approach to java synchronization. In: Proceedings of the 31st International Conference on Software Engineering, ICSE 2009, pp. 375–385, Washington (2009)
16. Cannon, B., Wohlstadter, E.: Enforcing security for desktop clients using authority aspects. In: Proceedings of the 8th ACM International Conference on Aspect-Oriented Software Development, AOSD 2009, pp. 255–266. New York (2009)
17. Malek, S., Krishnan, H.R., Srinivasan, J.: Enhancing middleware support for architecture-based development through compositional weaving of styles. J. Syst. Softw. **83**(12), 2513–2527 (2010)
18. Dyer, R., Rajan, H.: Supporting dynamic aspect-oriented features. ACM Trans. Softw. Eng. Methodol. **20**(2), 1–34 (2010)
19. Ortiz, G., Prado, A.G.D.: Improving device-aware web services and their mobile clients through an aspect-oriented model-driven approach. Inf. Softw. Technol. **52**(10), 1080–1093 (2010)

20. Toledo, R., Leger, P., Tanter, E.: AspectScript: expressive aspects for the web. In: Proceedings of the 9th International Conference on Aspect-Oriented Software Development, AOSD 2010, pp. 13–24 (2010)
21. Janik, A., Zielinski, K.: AAOP-based dynamically reconfigurable monitoring system. Inf. Softw. Technol. **52**(4), 380–396 (2010)
22. Ansaloni, D., Binder, W., Villazón, A., Moret, P.: Parallel dynamic analysis on multicores with aspect-oriented programming. In: Proceedings of the 9th International Conference on Aspect-Oriented Software Development, AOSD 2010, pp. 1–12 (2010)
23. de Roo, A., Sozer, H., Aksit, M.: Verification and analysis of domain-specific models of physical characteristics in embedded control software. Inf. Softw. Technol. **54**(12), 1432–1453 (2012)