doi:10.2498/cit.1001353

Examining the Relationships between Software Coupling and Software Performance: A Cross-platform Experiment

Liguo Yu[1] and Srini Ramaswamy[2]

[1] Computer and Information Sciences Department, Indiana University, South Bend, USA
[2] Industrial Software Systems, ABB Corporate Research Center, Bangalore, India

Coupling measures the degree of dependencies between software modules. Considerable research has been performed to relate software coupling with software understandability, maintainability, and reusability, which are the key properties of software maintenance and evolution. However, only a few research works have been reported that study the relationships between software coupling and software performance. This study implemented two benchmarks that measure and compare the performance of software programs implemented with different kinds of coupling, common coupling, data coupling, and stamp coupling. The experiment is run on three different platforms, Windows, Linux, and Mac. The results show that (1) the relative performance of systems implemented using different software coupling is platform dependent; (2) while loose coupling is more favorable than strong coupling with respect to software maintenance and evolution, it has the drawback of reduced performance of a software program. Based on this study, we make some suggestions to balance the use of strong coupling and loose coupling in designing evolving software systems in order to achieve both maintainability and evolvability without compromising on performance.

Keywords: component dependency, coupling, performance, cross-platform experiment

1. Introduction

Coupling is a measure of the degree of interaction between two software modules. Many different types of coupling have been identified, including data coupling, stamp coupling, control coupling, common coupling, and content coupling (Stevens et al., 1974; Page-Jones, 1980; Offutt et al., 1993, Offutt et al., 2008, Alexander et al., 2010). Table 1 lists the definitions of several major types of coupling in structured software systems, in which the degree of dependency is considered in increasing order from the bottom (data coupling) to the top (content coupling). Strong coupling means a high degree of dependency between software modules, while loose coupling means a low degree of dependency between software modules.

Name	Definition
Content Coupling	Two modules are content coupled if one accesses and changes the internal data or logic of the other.
Common Coupling	Two modules are common coupled if they refer to the same global variable.
Control Coupling	Two modules are control coupled if one passes a variable to the other that is used to control the internal logic of the other.
Stamp Coupling	Two modules are stamp coupled if they pass data through a parameter that is a record (structure).
Data Coupling	Two modules are data coupled if they pass data through a parameter that is a value.

Table 1. Definitions of various kinds of coupling in structured software systems (Offutt et al., 1993).

Maintenance and evolution are inevitable for a software product. After a software product is released, it requires continued maintenance and evolution. Coupling between software modules strengthens the dependency of one module on others and increases the probability that changes in one module may affect the other modules, which makes maintenance and evolution difficult and more likely to introduce regression faults (Banker et al., 1993; Schach et al., 2003; Yu et al., 2004). It has been shown that strong coupling is related to fault-proneness of a software system (Kafura and Henry, 1981; Selby and Basili, 1991; Troy and Zweben, 1981). Furthermore, the fault-proneness of one module can adversely affect the maintenance and evolution of a number of other modules. Therefore, many software engineering researchers have suggested that a good software system should have loose coupling (say, data coupling) between components while strong coupling (say, content coupling) should be avoided as much as possible.

Moreover, with the increase in the number of applications using distributed computing the concept of coupling is being extended beyond software modules contained within one program. It is now also used to represent interactions between software components in distributed systems. One such distributed computation framework is web services. Loose coupling or low dependencies between distributed software components indicate the flexibility of updating and reconfiguring services (Kaye, 2003; Erl, 2004). For example, if the server is down, loose coupling allows the client component to easily connect to a new server component and obtain the same service. Therefore, distributed computation also requires loose coupling between the client and the server components.

Hence, from the viewpoint of maintenance and evolution, both software modules within a system as well as software components distributed over a network should utilize loose coupling instead of strong coupling. However, the implementation differences between loose and strong coupling might have different effects on system performance. To our knowledge, very few research works have been done in this area, to understand the relationship between the issues of software coupling and software performance. This research is intended to study this issue by building two benchmarks to test the difference in performance of systems implemented with different types of coupling.

The remainder of this paper is organized as follows. Section 2 describes the research objectives. Section 3 describes the benchmarks and the experiment. Section 4 contains the results of the study. Conclusions and future work are in Section 5.

2. Research Objectives

For the different types of coupling listed in Table 1, control coupling and content coupling represent the control flow in the software program. Because of the poor design property of control coupling and content coupling, they are rarely used in both structured and object-oriented design methodologies. Especially content coupling is not supported by modern programming languages, such as C++ and Java. Therefore, this study will not investigate control coupling and content coupling.

The other three types of coupling identified in Table 1, common, stamp, and data coupling represent three different kinds of interactions between software modules. They are commonly used in most modern programming languages. Figure 1 describes the difference among these three types of coupling: the interactions between module m1 and module m2 can be implemented via common coupling (a), data coupling (b), or stamp coupling (c).

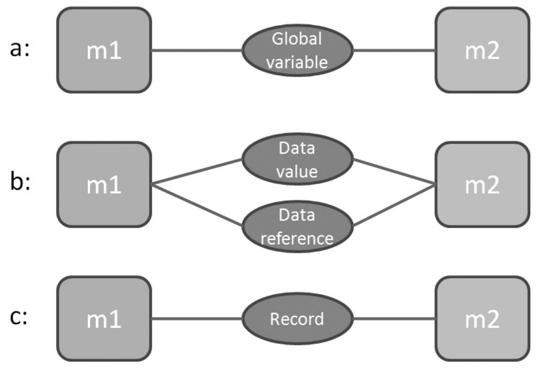

Figure 1. Three different types of coupling between module m1 and module m2: (a) common coupling; (b) data coupling; and (c) stamp coupling.

Common coupling is implemented through global variables. Because both m1 and m2 can access a global variable, the change of the value of the global variable done by one module could potentially affect another module. Data coupling between m1 and m2 is established through the function (method) call, in which either a data value or a data reference (data address or pointer) is passed between m1 and m2. Stamp coupling between m1 and m2 is achieved if a record (data structure, class) is passed between these modules. We remark here that stamp coupling could also be implemented through passing the reference (address, pointer) of a record. But it is not included in our current study.

In theory, any of these three types of coupling is not a unique choice between any two modules in a software system. In other words, the same effect of interactions between two modules can be implemented through either common, data, or stamp coupling. As we discussed before, data coupling is a much looser form of coupling than stamp coupling, which is in turn, a much looser form than common coupling. Hence, from the viewpoint of maintenance and evolution, we can extrapolate that data coupling is more favored over stamp coupling, which is more favored over common coupling. However, not much knowledge has been obtained with respect to the effect of these different types of coupling on software performance.

The objective of this study is to understand how data coupling (pass by data value and pass by data reference), stamp coupling (pass by data structure), and common (pass by global variable) might have different effects on the performance of a software system. The experiment is designed and run on different platforms to deeply understand such effects.

3. Benchmarks and Experiment Overview

The benchmarks introduced in this study implemented programs to verify the Goldbach's conjecture (Zenil, 2007). The reason we choose to verify Goldbach's conjecture is, first, the solution to this problem is pretty simple, few modules are needed to implement the benchmarks, which makes it easy to avoid the interference of other factors on our experiment. Second, appropriate number (not too few and not too many) of parameters are needed to pass between modules, which makes it both enough and easy to implement all three types of coupling. Basically, Goldbach's conjecture has two forms:

- **Conjecture 1:** Any even integer number greater than 2 can be written as the sum of two prime numbers.
- **Conjecture 2:** Any integer number greater than 5 can be written as the sum of three prime numbers.

In this study, Benchmark 1 implements a program to verify Conjecture 1, and Benchmark 2 implements a program to verify Conjecture 2. The architecture of the benchmarks is shown in Figure 2. In each benchmark, three different interactions are implemented between the **Main control** module and the **Find primes** module. In Benchmark 1, the three implementation are common coupling (pass by global variable (GV)), data coupling (pass by data value (DV)), and data coupling (pass by data reference (DR)). In Benchmark 2, the three implementations are common coupling (GV), data coupling (DR), and stamp coupling (pass by data structure – (DS)). The **Main control** modules can choose any of the three implementations to run the experiment.

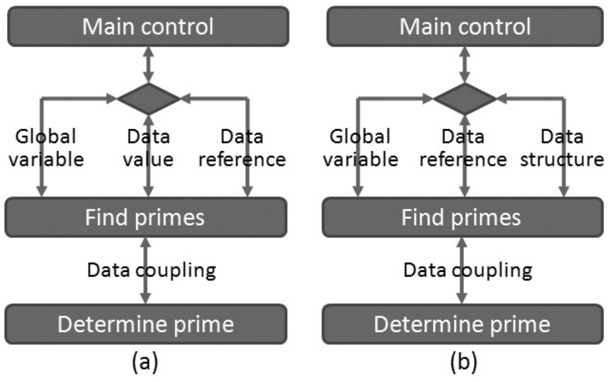

Figure 2. The architecture of the benchmarks: (a) Benchmark 1; and (b) Benchmark 2.

The benchmarks are implemented using C/C++ for the purpose of cross-platform compatibility. Three different platforms (Windows, Linux, and Mac) are chosen to run the benchmarks. The detail information about the three platforms is listed in Table 2.

Platform #	Hardware Vendor	Operating System	Compiler
1	Dell	Windows XP	Visual Studio
2	Dell	Red Hat Beowulf	GCC
3	Apple	Mac	GCC

Table 2. The experiment platforms.

The scenarios of running the two benchmarks are listed in Table 3. Benchmark 1 verifies Conjecture 1 using even numbers in the specified range of different scenarios; Benchmark 2 verifies Conjecture 2 using all numbers in the specified range of different scenarios. Each scenario for each benchmark is run 100 times for each platform. The *interaction time* (time elapsed between sending the request and receiving the result between the **Main control** module and the **Find primes** module) is recorded in milliseconds. Therefore, with 54 experiments, each is run 100 times, a total of 5400 data points were collected.

	Benchmark 1	Benchmark 2
Scenario 1	Even number between [4, 10000]	Any number between [6, 1000]
Scenario 2	Even number between [4, 20000]	Any number between [6, 2000]
Scenario 3	Even number between [4, 40000]	Any number between [6, 4000]

Table 3. Scenarios of running the benchmarks.

4. Experiment Results

4.1. Data Stability Analysis

Before the data are finely analysed, we study the stability of the data collected. To validate the data of 54 experiments, the mean and the standard deviation of the interaction time collected in 100 iterations are calculated. The *experiment stability* is defined using the following formula.

Definition 1: *experiment stability=standard deviation/mean*

The *experiment stability* metric represents the average percentage difference of each measurement to the mean of all 100 measurements. Table 4 shows the *Experiment Stability* of all 54 experiments.

From Table 4, we see that all the stability measurements have values less than 8%. This means the systems are pretty stable. This analysis validates data stability and indicates that our experiments are not affected by other programs that might run at the same time.

4.2. Detailed analysis

4.2.1. Benchmark 1

Table 5 through Table 7 shows the detailed results of running Benchmark 1 on Windows, Linux, and Mac respectively. For all three scenarios running on three platforms, similar results were obtained: no big differences were found among the mean of the interaction times of three different types of coupling, pass by global variable (common coupling), pass by data value (data coupling), and pass by data reference (data coupling).

Platform		Benchmark 1			Benchmark 2		
	Pass by	GV	DV	DR	GV	DR	DS
Windows	Scenario 1	0.32%	2.67%	0.36%	3.33%	0.51%	0.56%
	Scenario 2	0.13%	0.80%	0.35%	6.74%	0.32%	1.10%
	Scenario 3	0.12%	0.42%	1.15%	1.66%	0.21%	0.21%
Linux	Scenario 1	0.95%	5.43%	3.96%	$< 0.01\%$	$< 0.01\%$	1.48%
	Scenario 2	0.43%	5.21%	2.77%	$< 0.01\%$	$< 0.01\%$	0.82%
	Scenario 3	1.59%	7.69%	0.86%	1.80%	1.76%	0.30%
Mac	Scenario 1	0.21%	0.63%	0.25%	$< 0.01\%$	$< 0.01\%$	$< 0.01\%$
	Scenario 2	0.20%	0.20%	0.21%	$< 0.01\%$	$< 0.01\%$	$< 0.01\%$
	Scenario 3	0.17%	0.19%	0.18%	$< 0.01\%$	0.98%	1.50%
GV: global variable; DV: data value; DR: data reference; DS: data structure							

Table 4. The Experiment Stability.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	2.484	2.515	2.501
Pass by data value	2.484	3.031	2.512
Pass by data reference	2.484	2.515	2.495

Table 5a. Measurement of Benchmark 1, Scenario 1 on Windows.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	10.828	10.953	10.838
Pass by data value	10.807	11.453	10.859
Pass by data reference	10.828	11.109	10.845

Table 5b. Measurement of Benchmark 1, Scenario 2 on Windows.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global data variable	46.734	47.281	46.787
Pass by data value	46.658	48.126	46.721
Pass by data reference	46.673	48.857	46.960

Table 5c. Measurement of Benchmark 1, Scenario 3 on Windows.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	2.10	2.26	2.114
Pass by data value	2.11	2.86	2.183
Pass by data reference	2.11	2.56	2.146

Table 6a. Measurement of Benchmark 1, Scenario 1 on Linux.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	9.15	9.34	9.180
Pass by data value	9.17	11.59	9.387
Pass by data reference	9.15	10.44	9.278

Table 6b. Measurement of Benchmark 1, Scenario 2 on Linux.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	39.45	44.30	39.697
Pass by data value	39.44	52.22	39.739
Pass by data reference	39.46	42.69	39.684

Table 6c. Measurement of Benchmark 1, Scenario 3 on Linux.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	2.23	2.25	2.24
Pass by data value	2.23	2.28	2.247
Pass by data reference	2.23	2.25	2.239

Table 7a. Measurement of Benchmark 1, Scenario 1 on Mac.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	9.72	9.77	9.747
Pass by data value	9.72	9.78	9.741
Pass by data reference	9.72	9.78	9.737

Table 7b. Measurement of Benchmark 1, Scenario 2 on Mac.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	42.07	42.25	42.144
Pass by data value	41.99	42.19	42.041
Pass by data reference	41.09	42.16	41.992

Table 7c. Measurement of Benchmark 1, Scenario 3 on Mac.

The detailed analyses are shown in Figure 3 and Figure 4. Figure 3 shows the *performance difference* of three measurements (pass by global variable, pass by data value, and pass by data reference). *Performance difference* is defined using the following formula.

Definition 2: *performance difference=mean of each measurement/average of the means of three measurements*

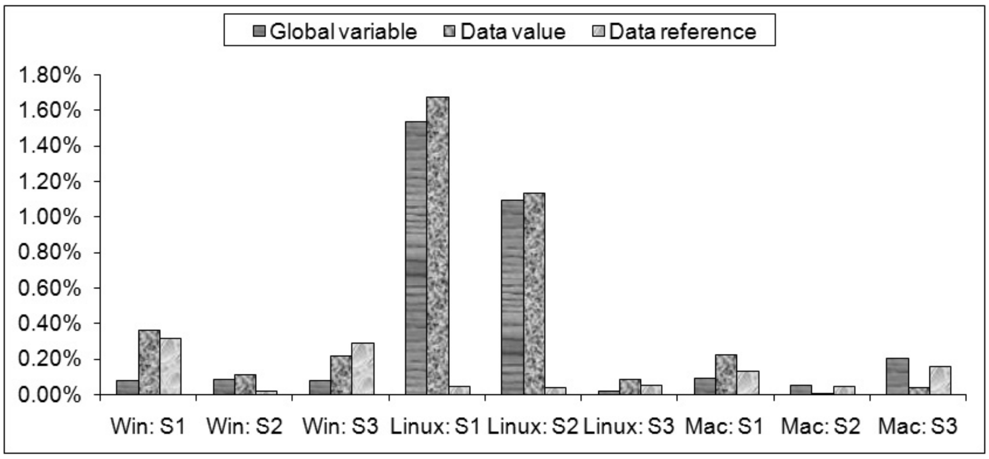

Figure 3. The performance difference of three implementations of couplings (pass by global variable, pass by data value, pass by data reference) measured in three scenarios (S1, S2, and S3) running on three platforms (Win, Linux, and Mac).

The performance difference could be significant or insignificant. In this work, we consider a difference of 10% or above as *significant*, while less than 10% difference as *insignificant*.

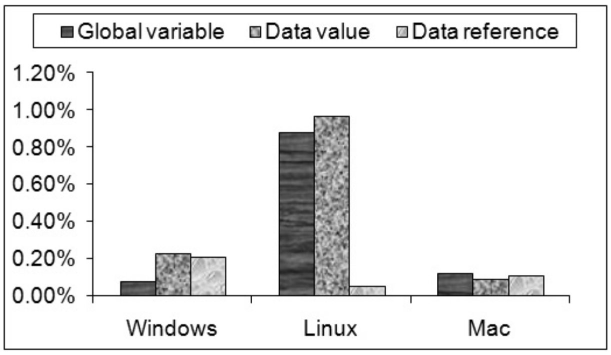

Figure 4. The average performance differences of three implementations of coupling (pass by global variable, pass by data value, and pass by data reference).

Figure 4 averages the *performance difference* of the three scenarios with respect to each platform. It can be seen that in Windows and Mac, the average *performance differences* of three implementations of coupling are less than 0.5%, while in Linux, the average *performance difference* of three implementations of coupling are less than 2.5%. Therefore, based on the Benchmark 1, we conclude that there are no significant performance differences for coupling implemented via pass by global variable (common coupling), pass by data value (data coupling), or pass by data reference (data coupling). Comparing the three platforms, we see that experiments running on three different platforms return similar results. However, we also notice in Figure 3 and Figure 4, Linux has higher performance difference than Windows and Mac. So, we conclude that Benchmark 1 is platform dependent, but not highly sensitive. Here we use the word *dependent* to represent any difference (*insignificant* or *significant*) and the word *sensitive* to represent a significant difference.

4.2.2. Benchmark 2

Table 8 through Table 10 show the detailed results of running Benchmark 2 on Windows, Linux, and Mac respectively. Results show that three different scenarios running on the same platform return similar results: For windows, pass by global variable (common coupling) has much smaller interaction time than pass by data reference (data coupling) and pass by data structure (stamp coupling); For Linux: pass by global variable (common coupling) and pass by data reference (data coupling) has much smaller interaction time than pass by data structure (stamp coupling); For Mac: no big differences of interaction time are found among pass by global variable (common coupling), pass by data reference (data coupling), and pass by data structure (stamp coupling).

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	0.015	0.031	0.018
Pass by data reference	1.374	1.421	1.386
Pass by data structure	1.374	1.422	1.389

Table 8a. Measurement of Benchmark 2, Scenario 1 on Windows.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	0.078	0.109	0.089
Pass by data reference	11.040	11.384	11.059
Pass by data structure	11.065	12.299	11.100

Table 8b. Measurement of Benchmark 2, Scenario 2 on Windows.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	0.359	0.374	0.362
Pass by data reference	87.950	89.324	88.010
Pass by data structure	87.640	89.254	88.035

Table 8c. Measurement of Benchmark 2, Scenario 3 on Windows.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	0.009	0.009	0.009
Pass by data reference	0.009	0.009	0.009
Pass by data structure	1.110	2.259	1.114

Table 9a. Measurement of Benchmark 2, Scenario 1 on Linux.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	0.061	0.061	0.061
Pass by data reference	0.061	0.061	0.061
Pass by data structure	8.933	9.409	8.987

Table 9b. Measurement of Benchmark 2, Scenario 2 on Linux.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	0.282	0.299	0.285
Pass by data reference	0.282	0.308	0.288
Pass by data structure	71.586	72.537	71.996

Table 9c. Measurement of Benchmark 2, Scenario 3 on Linux.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	0.01	0.01	0.01
Pass by data reference	0.01	0.01	0.01
Pass by data structure	0.01	0.01	0.01

Table 10a. Measurement of Benchmark 2, Scenario 1 on Mac.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	0.07	0.07	0.07
Pass by data reference	0.07	0.07	0.07
Pass by data structure	0.07	0.07	0.07

Table 10b. Measurement of Benchmark 2, Scenario 2 on Mac.

Coupling type	Interaction time (seconds)		
	Minimum	Maximum	Mean
Pass by global variable	0.32	0.33	0.32
Pass by data reference	0.32	0.33	0.321
Pass by data structure	0.32	0.33	0.323

Table 10c. Measurement of Benchmark 2, Scenario 3 on Mac.

The detailed analysis is given in Figure 5, which shows the performance difference of three measurements (pass by global variable, pass by data reference and pass by data structure), and in Figure 6, which averages the performance difference of three scenarios with respect to each platform. It is worth noting that the performance differences in Mac are less than 0.52% in all scenarios.

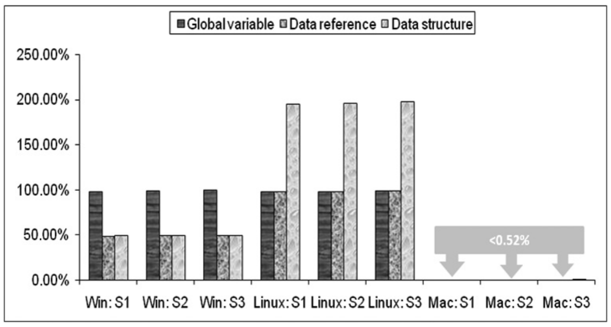

Figure 5. The performance difference of three implementations of coupling (pass by global variable, pass by data reference, pass by data structure) measured in three scenarios (S1, S2, and S3) running on three platforms (Win, Linux, and Mac).

It can be seen that although insignificant differences of performance of Benchmark 2 are found in Mac, they are found in Windows and Linux, where over 50% of performance differences are detected. Therefore, based on the Benchmark 2, we conclude that there are significant performance differences for coupling implemented via pass by global variable (common coupling), pass by data reference (data coupling), and pass by data structure (data stamp coupling). It also tells us that the experiments running on three different platforms return different results, which indicates that Benchmark 2 is platform dependent and sensitive.

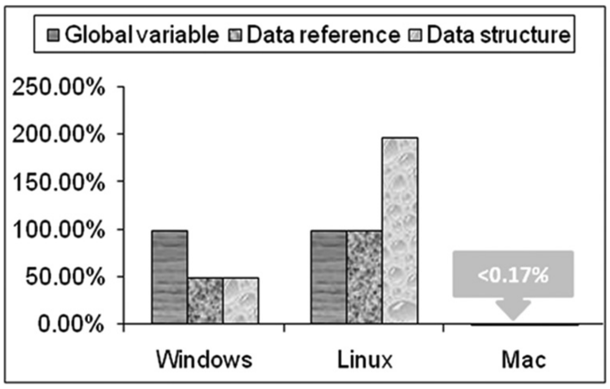

Figure 6. The average performance differences of three implementations of coupling (pass by global variable, pass by data reference, and pass by data structure).

4.2.3. Discussion

Based on the experiments of two benchmarks on three platforms, we found that Benchmark 1 shows no significant performance differences of pass by global variable, pass by data value, and pass by data reference. However, Benchmark 2 returns significant difference among these couplings in Windows and in Linux. Specifically, we found, (1) in Windows, pass by global variable (common coupling) has less interaction time than pass by data reference (data coupling), and pass by data structure (stamp coupling); and (2) in Linux, pass by global variable (common coupling) and pass by data reference (data coupling) have less interaction time than pass by data structure (stamp coupling).

On one hand, pass by global variable (common coupling) and pass by data reference (data coupling) do not involve the reallocation of memory as is done in pass by data value (data coupling), and, accordingly, we expected them to have less interaction time. On the other hand, global variable and local variable might be allocated on different locations of memory (heap or stack), the access time difference to these locations might affect the interaction time of pass by global variable and pass by data references. While these differences are not observed in Benchmark 1, some noticeable differences are observed in Benchmark 2.

Stamp coupling is related with passing values of a data structure. Because a data structure (not a reference to a data structure, but the data structure itself) is passed between two modules, reallocation of memory is involved in this process. Therefore, it is expected to have more interaction time than pass by global variable (common coupling) and pass by data reference (data coupling). These differences are observed in Benchmark 2. Figure 7 summarizes all of our observations and analyses. In all of the

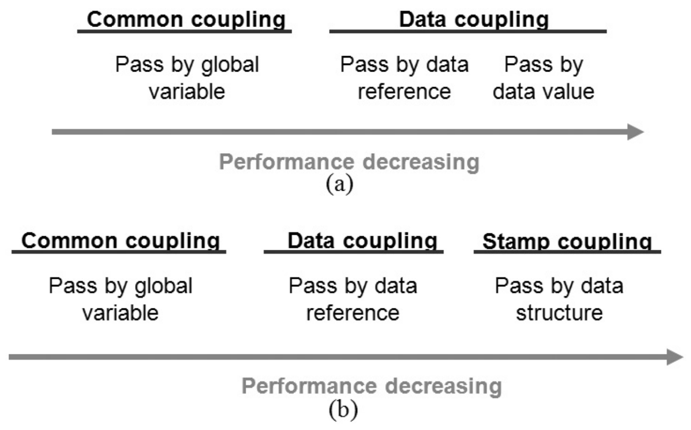

Figure 7. The performance differences of software modules implemented via different couplings.

experiments, pass by global variable (common coupling) has the smallest interaction time.

It should be mentioned that these observed differences are not platform independent. For some platforms, the differences are significant, for others, the differences are insignificant. We speculate that the behavior differences of different platforms are due to either the design of the operating system (such as resource allocation (ex. memory) mechanism), or the hardware structure (the addressing and accessing mechanism). More experiments are needed to understand the effects of the platforms on software performance.

As described before, strong coupling (such as common coupling) has proven drawbacks of decreasing software understandability, maintainability, and evolvability. Hence, it is generally recommended to be avoided as much as possible. Loose coupling, such as parameter passing through data values or data structures, is more beneficial to software maintenance and evolution. However, our results through this study show that strong coupling, such as common coupling, may lead to higher performance over loose coupling. Therefore, to compensate the contradictory effect of coupling in balancing between software maintenance needs versus software performance needs, we make the following suggestions.

1. The use of strong coupling and loose coupling should be appropriately balanced in designing evolving software systems in order to achieve both maintainability and evolvability without compromising obtaining a higher level of performance.
2. While maintainability and evolvability of software programs are platform independent, the performance of software programs is more platform dependent. In software performance testing, exclusive tests need to be performed on different platforms to verify the effects of different types of coupling on the software system's performance.

5. Conclusions

In this paper, we studied the performance difference of software systems that provide same functionality, but is implemented utilizing different types of coupling. We built two benchmarks and ran these experimentations on three different widely-used platforms; Windows, Linux, and Mac. Our results indicate that the performance of different types of coupling is platform dependent. Furthermore, we identified that, in general, strong coupling leads to higher performance than loose coupling, with respect to the interaction time between software modules.

Future work will extend current study in other areas of software performance, with respect to software coupling and software dependency. The specific plan is listed below.

1. We will study other types of coupling and their relations with software performance. More specifically, we will study stamp coupling, which is induced through pass by reference of a data structure in structured system and inheritance coupling that occurs in an object-oriented system.

2. We will study in more depth the platform dependency property of software performance, to understand how coupling is related to different platforms (operating systems, hardware), which might also affect software performance.
3. We will study the performance differences in interactions between components located on distributed networks. For example, in service-oriented computation, the coupling type between client component and service component might affect their interaction efficiency.

References

[1] R. T. ALEXANDER, J. OFFUTT, A. STEFIK, Testing Coupling Relationships in Object-oriented Programs. *Software Testing, Verification & Reliability*, vol. 20, no. 4 (2010), pp. 291–327.

[2] R. D. BANKER, S. M. DATAR, C. F. KEMERER, D. ZWEIG, Software Complexity and Maintenance Costs. *Communications of the ACM*, vol. 36, no. 11 (1993), pp. 81–94.

[3] T. ERL, *Service Oriented Architecture: A field Guide to Integrating XML and Web Services*. Prentice Hall PTR, April 2004.

[4] D. KAFURA, S. HENRY, Software Quality Metrics Based on Interconnectivity. *Journal of Systems and Software*, vol. 2, no. 2 (1981), pp. 121–131.

[5] D. KAYE, *Loosely Coupled: The Missing Pieces of Web Services*. RDS Associates; 1st edition, August 2003.

[6] J. OFFUTT, M. J. HARROLD, P. KOLTE, A Software Metric System for Module Coupling. *Journal of Systems. and Software*, vol. 20, no. 3 (1993), pp. 295–308.

[7] J. OFFUTT, A. ABDURAZIK, S. R. SCHACH, Quantitatively Measuring Object-oriented Couplings. *Software Quality Journal*, vol. 16, no. 4 (2008), pp. 489–512.

[8] M. PAGE-JONES, *The Practical Guide to Structured Systems Design*. Yourdon Press, New York, 1980.

[9] S. R. SCHACH, B. JIN, D. R. WRIGHT, G. Z. HELLER, J. OFFUTT, Quality Impacts of Clandestine Common Coupling. *Software Quality Journal*, vol. 11 (2003), pp. 211–218.

[10] R. W. SELBY, V. R. BASILI, Analyzing Error-prone System Structure. *IEEE Transactions on Software Engineering*, vol. 17, no. 2 (1991), pp. 141–152.

[11] W. P. STEVENS, G. J. MYERS, L. L. CONSTANTINE, Structured Design. *IBM Systems Journal*, vol. 13, no. 2 (1974), pp. 115–139.

[12] D. A. TROY, S. H. ZWEBEN, Measuring the Quality of Structured Design. *Journal of Systems and Software*, vol. 2, no. 2 (1981), pp. 113–120.

[13] L. YU, S. R. SCHACH, K. CHEN, J. OFFUTT, Categorization of Common Coupling and its Application to the Maintainability of the Linux Kernel. *IEEE Transactions on Software Engineering*, vol. 30, no. 10 (2004), pp. 694–706.

[14] H. ZENIL, Goldbach's Conjecture. The Wolfram Demonstrations Project, 2007, `http://demonstrations.wolfram.com/GoldbachConjecture/`

Received: September, 2008
Accepted: March, 2011

Contact addresses:

Liguo Yu
Computer and Information Sciences Department
Indiana University South Bend
1700 Mishawaka Ave.
P.O. Box 7111
South Bend, IN 46634, USA
e-mail: `ligyu@iusb.edu`

Srini Ramaswamy
Industrial Software Systems
ABB Corporate Research Center
Bangalore, India
e-mail: `srini@ieee.org`

LIGUO YU is an assistant professor at Computer Science Department, Indiana University South Bend. He received his PhD degree in computer science from the Vanderbilt University in 2004 and his MS degree from the Institute of Metal Research, Chinese Academy of Science in 1995 and his BS degree in physics from the Jilin University in 1992. Before joining IUSB, he was a visiting assistant professor at Tennessee Tech University. His research interests include software dependency, software maintenance, software reuse, software evolution, empirical software engineering and open-source development.

SRINI RAMASWAMY earned his Ph.D. degree in computer science in 1994 from the Center for Advanced Computer Studies (CACS) at the University of Southwestern Louisiana (now University of Louisiana at Lafayette). His research interests are in intelligent and flexible control systems, behavior modeling, analysis and simulation, software stability and scalability. Currently, he is the head of Industrial Software Systems, ABB Corporate Research Center, Bangalore, India. Before joining ABB, he was the Chairperson of the Department of Computer Science, University of Arkansas at Little Rock and the chairman of Computer Science Department at Tennessee Tech University. He is a member of the ACM, SCSI, CPSR, and a senior member of the IEEE.