

A Study on the Most Popular Questions about Concurrent Programming

Gustavo Pinto, Wesley Torres, and Fernando Castor

Federal University of Pernambuco, Recife, Brazil

{ghlp,wst,castor}@cin.ufpe.br

Abstract

Concurrent programming is notably known as a hard discipline. Over the last few years, great strides have been made in improving concurrent programming abstractions, techniques, and tools to ease concurrent programming practice. However, little effort has been placed on assessing what are the real-world problems faced by developers when writing concurrent applications. In this paper, we describe an empirical investigation of the top-250 most popular questions about concurrent programming on STACKOVERFLOW. We observed that even though some questions (22.94%) are related to practical problems (e.g., “how to fix this concurrency bug”), most of them (66.23%) are related to basic concepts (e.g., “what is a mutex?”), which were created by well-experienced STACKOVERFLOW users. Curiously, we did not find any question about how to use concurrent programming techniques to improve application performance.

Categories and Subject Descriptors H.4 [Information Systems Applications]: Miscellaneous

General Terms Documentation, Human Factors

Keywords Concurrency, Practitioners, Q&A

1. Introduction

Although the idea of concurrency and parallelism has been around since the first computer [27], multicore processors became mainstream roughly a decade ago, when AMD and Intel started selling dual core processors for desktops. To better leverage multicore technology, applications must be concurrent, which poses a challenge, since it is well-known that concurrent programming is hard [15, 26]. To mitigate

the burden of concurrent programming, great strides have been made in introducing new concurrent programming frameworks [13], models [10], and also empirical studies on, for instance, how developers use/misuse concurrent programming constructs [17, 23], as well as performance [2], programmer effort, satisfaction and error-proneness [19] and even energy consumption [22] evaluations.

Even though several studies have been conducted on the concurrent programming arena [2, 10, 22], to the best of our knowledge, there is a lack of studies targeting what are the real-world problems faced by application developers when writing concurrent applications. Although Lu *et al.* [14] provided a comprehensive taxonomy of concurrency bugs, the authors used a high-quality number of bug reports as their dataset, which are mostly created by advanced programmers [7]. In this study, however, we focus on STACKOVERFLOW. As one of the most popular website about software development, STACKOVERFLOW, instead a place for experts only, contains a diverse range of software developers [16].

This critical direction deserves more investigation due to at least two reasons: (1) without an understanding of the needs of developers and the challenges they face, researchers may be naively focused on solving problems that, albeit interesting, are faced by few developers in practice; and (2) the conceptually incorrect views they hold may inspire educators to develop more state-of-the-art curricula. In particular, we are interested in the following research questions:

- RQ1** What are the most popular problems faced by software developers when writing concurrent software?
- RQ2** What are the concurrent programming concepts that software developers want to know more about?
- RQ3** What are the unanswered questions about concurrent programming?

To answer these RQs, we provide a comprehensive list of the top-250 most popular questions about concurrency asked in STACKOVERFLOW. We chose the top most popular questions instead of, for instance, a random sample, because these questions are more often visualized (by both registered and non-registered STACKOVERFLOW users). Therefore, we believe that we can reach a larger number of real-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

PLATEAU'15, October 26, 2015, Pittsburgh, PA, USA
© 2015 ACM. 978-1-4503-3907-0/15/10...\$15.00
<http://dx.doi.org/10.1145/2846680.2846687>

world programmers interested in concurrency. Moreover, the popularity of the questions can be seen as a proxy for the popularity of concurrency-related problems. As mentioned elsewhere, these questions capture a problem that developers experienced [11].

The main findings of this study are the following:

- Our group of questions have 4.99x more answers, are marked as favorites 21.49x more often, and have 41.97x more views, 62.33x more “up-votes”, and 1.43x more comments than the average questions on STACKOVERFLOW. Most of our questions (59.74% of them) and their answers (44.50% of them) are old (from 2008 to 2009). STACKOVERFLOW users who asked such questions are well-experienced rather than novice programmers.
- We observed that most of questions are about well-established concepts. Threading and synchronization issues are the most common ones. Advanced concepts, such as software transactional memory, are never mentioned. Also, albeit useful for the game industry, GPU programming was mentioned by only one question.
- We found out that, even though all questions have answers, some questions do not have an *accepted* answer. Interestingly, only the minority of them (17.85%) require an in-depth concurrent programming knowledge. The remaining ones do have comprehensive candidates of accepted answers, but the STACKOVERFLOW user who created the questions did not mark any answer as accepted.

2. Methodology

Our data collection follows a mixed-method approach, collecting both quantitative and qualitative data. Using the STACKEXCHANGE¹ website, we are able to run queries in the most up-to-date STACKOVERFLOW database. Using this database, we extracted questions, answers, tags, and other metadata. The data reported in this paper are based on questions that were asked between August 15, 2008 and October 6, 2014. The first date is related to when STACKOVERFLOW was first launched, and the second one was when we ran the query. An initial search reveals that the “concurrency” topic has more than 22,000 questions. This fact prevents a manual analysis from being successful.

Instead, we decided to carefully select and analyze the top-250 most popular questions — that is, about 1% of the overall concurrency-related questions. In order to effectively identify this curated list, we first queried the questions that have at least one concurrency-related keyword among its tags. We did not filter in the body or in the title because of two reasons: (1) every question has at least one tag, and (2) related questions will, with a high probability, use a tag that is self-evident. We used the same set of keywords used by Lu *et al.* [14], which encompasses “race(s)”,

“deadlock(s)”, “livelock(s)”, “concurrency”, “lock(s)”, “mutex(es)”, “atomic”, “compete(s)”, and “multithreading”. Finally, from the thousands of questions that contain at least one keyword from the above keyword set, we filtered the top-250 by using a popularity metric similar to one we employed in previous work [21], with some modifications:

$$\mathbb{P} = \text{GMEAN}(\mathbb{S}, \mathbb{A}, \mathbb{C}, \mathbb{F}, \mathbb{V})$$

where \mathbb{S} is the score (up-votes and down-votes) of the question. The variables \mathbb{A} , \mathbb{C} , \mathbb{F} and \mathbb{V} are, respectively, the number of answers, comments, favorizations, and the number of views per question. We normalize each variable to avoid distortions caused by very large absolute values. Taking the \mathbb{V} variable as an example, it is normalized by the average of the number of views of the overall STACKOVERFLOW questions, as follows: $\mathbb{V} = \text{questionsViews} / \text{stackOverflowViews}$. The other variables follow the same rule. The value of \mathbb{P} is calculated using a geometric mean (the GMEAN function) of the other variables. According to Fleming and Wallace [4], the use of geometric mean is favorable than an average.

After this automatic process, we manually checked them to make sure that they are related to concurrent programming. During this process, we found and removed 19 false-positive questions. Example of such questions are question Q309396², which asks “*Java: How to test methods that call System.exit()?* ”, and question Q991904, which asks “*Why is there no GIL in the Java Virtual Machine? Why does Python need one so bad?*”³.

Hereafter we call this group of questions as our **base group**, which encompasses 231 questions. Once the data is collected, we extract reliable information using a thematic analysis [3]. It has six steps:

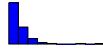
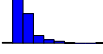
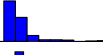
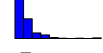

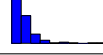
1. *Familiarization with data*: At this stage, we analyzed our base group of questions and their answers. When a STACKOVERFLOW user mentioned very specific construct or library, we have searched on internet forums and in mailing lists in order to better familiarize with them.
2. *Generating initial codes*: Here we gave a code for each question. This code tries to summarize the core of the question. A question that asks for how to stop a thread was coded as “Thread Life-Cycle”. We also refine codes by combining and splitting potential codes.
3. *Searching for themes*: In this step, we already had a list of initial themes (*e.g.*, threading and synchronization), but we begin to focus on broader patterns in the data, combining coded data with proposed themes.

²The “Q309396” notation refers to the id of the question in STACKOVERFLOW, which can be translated to <http://stackoverflow.com/questions/309396> url. We use this notation throughout the paper.

³For most cases, we kept the title of the question *ipsis litteris*. We added the words in square brackets, though.

¹<http://data.stackexchange.com/>

Table 1. The popularity of questions in each group of data.

#	Base Group	Median	Std.	Histogram
S	62.39	43.13	63.15	
A	5.05	4.07	3.3	
C	1.43	0.6	2.12	
F	21.34	13.98	23.34	
V	42.14	30.86	39.7	
P	13.22	90.32	113.4	

4. *Reviewing themes:* Here we have a potential set of themes. We then searched for data that supports or refutes our themes. For instance, we initially themed the question “How to wait for thread to finish with .NET??” as “Practical Concepts”. However, we later realized that this question would fit better as “Thread Life-Cycle”.
5. *Defining and naming themes:* Here we refined existing themes. At this time, most of the themes had a name. We renamed some of them to cover codes with few questions, since we established 5 as the minimum number of repetitions required to code be considered a theme.
6. *Producing the final report:* This process leads to 7 main themes. We discuss each one in § 3.1.

3. Research Results

Before we discuss each research question, we provide quantitative aspects regarding our base group of questions. First, we have compared the popularity of concurrent programming questions with the remaining STACKOVERFLOW questions. We computed the popularity using the metric described in the last section. Table 1 shows the results.

Since the value for each metric of the STACKOVERFLOW group is normalized to be 1 (thus the geometric mean \mathbb{P} for $\mathbb{S}=1$, $\mathbb{A}=1$, $\mathbb{C}=1$, $\mathbb{F}=1$, $\mathbb{V}=1$ is also 1) we can observe that questions from our base group are more than 13.22x more popular when compared to the remaining questions on STACKOVERFLOW. Among the individual variables, the \mathbb{C} variable is the only one which does not present a great difference; it is only 1.43 times greater than the remaining questions on STACKOVERFLOW. This fact is due to 88 questions that do not have a single comment. Since comments on questions are mostly used to ask for clarifications, we can assume that questions from the base group are *well-elaborated*. This can also help us explain the great number of answer per question (\mathbb{A} is 5.05). All other variables present a considerable difference. We observed that the score of a question (the \mathbb{S} variable) presents the highest difference when compared to the remaining questions on STACKOVERFLOW (\mathbb{S} max:

540.62, min: 46.87, 3rd quartile: 160). It means that more STACKOVERFLOW users are “upvoting” those questions.

Most of the questions in our base group were created between 2008 and 2009 (59.74% of them). We consider these questions as old. This is not surprising. Once a question has been answered, the community tries to avoid duplication by pointing to the older question when new users ask it again. From the 2,146 answers to these questions, 44.50% are old.

As regarding the answers of those questions, we calculated their popularity as $\mathbb{P}\mathbb{A} = \text{GMEAN}(\mathbb{S}, \mathbb{C})$. We observed that comments presented a slight difference against the remaining answers on STACKOVERFLOW (\mathbb{C} is 1.15), whereas the score of these answers is 14.07x greater than the remaining answers on STACKOVERFLOW. $\mathbb{P}\mathbb{A}$ is 4.02.

3.1 RQ1. The most popular problems

In this RQ, we analyzed the titles and bodies of our base group of questions. We grouped them into 7 main themes, each described below. After each one, we list three questions to illustrate the diversity of their interests.

Theoretical Concepts This theme groups questions where the STACKOVERFLOW user would like to understand a concurrent programming concept. We found a total of 55 questions asked (6 of them are in the top-10 most popular ones⁴). These questions fit into one of the two following templates: (1) what is X? or (2) what is the difference between X and Y? For instance:

Q34510 ~ “What is a race condition?”

Q588866★ ~ “What’s the difference between the atomic and nonatomic attributes?”

Q200469★ ~ “What is the difference between a process and a thread?”

Since this theme groups questions that ask for general guidelines, we believe that the trade-off between techniques should be better explained. We found 29 questions in this theme where a STACKOVERFLOW user asked about when to use one techniques instead of another (e.g., Q7889746, Q800383, Q4201713). As a STACKOVERFLOW user mentioned “I got slightly confused about the differences between *Handlers, AsyncTasks and Threads in Android*. ... Should this be run in a handler or a thread, or even an async task?” (Q6964011). API and language designers can dramatically improve this scenario by providing general explanations on whether or not to favor one concurrent programming construct instead of another. Also, researchers can create recommendation systems that support developers in better exploring these trade-offs. We categorize the questions in this theme into sub-themes in § 3.2.1.

Practical Concepts Unlike the “teorical concepts”, in this theme, STACKOVERFLOW users already understand a certain concurrent programming construct or concept, but they

⁴ The symbol ★ denotes that the question is part of the top-10 most popular ones.

do not know in which situations they should use them (51 questions asked — 2 of them are in the top-10 most popular ones). We then grouped questions that asked for (1) how to use X and (2) when to use X. For instance:

Q817856★ ~ “When and how should I use a ThreadLocal variable?”

Q154551 ~ “Which [synchronization] approach should be used [to increment an int]?”

Q251391 ~ “Why is lock(this) {...} bad?”

Here we also found 16 questions comparing two different concurrent programming constructs. Still, STACKOVERFLOW users are highly interested in best practices (e.g., Q251391) and practical uses (e.g., Q4818699). Interestingly, we found 5 questions regarding Java’s volatile keyword. We observed that, even though STACKOVERFLOW users know what is this concurrent construct for, they are not sure when it can be an appropriate replacement for a synchronized statement, which provides *more* guarantees, or for an AtomicReference (e.g., Q281132), which provides the *same* guarantees [6]. We categorize the questions in this theme into sub-themes in § 3.2.2.

First Steps We observed that STACKOVERFLOW users are looking for the simplest working multithreaded code (29 questions asked). In this theme we grouped questions that asked for (1) how to create a thread?, (2) does language X supports threads?, and also (3) questions about thread usage such as (3.1) how to pass a parameter to a thread?, (3.2) how to get the number of processors available?, or (3.3) how to get a thread’s id? Some examples include:

Q2846653 ~ “Trying to find a simple example that clearly shows a single task being divided for multi-threading.”

Q3360555 ~ “How to pass parameters to ThreadStart method in Thread?”

Q2734025 ~ “Is javascript guaranteed to be single-threaded?”

We found several questions asking for the simplest multithreaded working code, either for C++ (Q266168), Android (Q1921514), or even PHP (Q70855). Since these questions are relatively easy to answer, all of them were solved. Another common category of questions is related to managing objects in different threads, for instance, how to pass a parameter to a thread (e.g., Q877096), or how to return a value from a thread (e.g., Q1314155). Best practices are also encouraged here, as a STACKOVERFLOW user mentioned, “[I] just want some advice on ‘best practice’ regarding multi-threading tasks.” Q2528907.

Thread Life-Cycle Several STACKOVERFLOW users asked about the thread life-cycle (18 questions asked — 1 of them is in the top-10 most popular ones). In this theme, we grouped the questions related to (1) how to join/wait/sleep/interrupt/terminate⁵ a thread. For instance:

⁵ All questions related to thread creation are in the “First Steps” theme. We opted for this approach because there is a number of issues related to thread creations, for instance, how to pass different parameters to a thread Q877096.

Q1520887 ~ “How to pause / sleep thread or process in Android?”

Q323972 ~ “Is there any way to kill a Thread in Python?”

Q289434 ~ “How to make a Java thread wait for another thread’s output?”

Here questions ask about how to wait/join/pause/sleep/stop a thread. Interestingly, 8 (out of 18 questions) pertain to how to stop a thread. However, it is generally a bad pattern to try to stop a thread. In Java, where the method Thread.stop() is deprecated, the documentation is clear in saying that it is an “*inherently unsafe*” way to stop a thread⁶, because it can create objects with inconsistent states. A better approach is to use the Thread.interrupt() method instead. STACKOVERFLOW users are aware of that. All answers have mentioned that the Thread.interrupt() method is the proper way to solve this problem.

Technical problems This theme is related to technical problems faced by developers when writing concurrent applications (53 questions asked — 1 of them is in the top-10 most popular ones). We grouped questions that asked for (1) how to fix this problem? For instance:

Q661561★ ~ “How to update the GUI from another thread in C#?”

Q530211 ~ “[How to] create a blocking Queue<T> in .NET?”

Q16159203 ~ “Why does this Java program terminate despite that apparently it shouldn’t (and didn’t)?”

In this group of questions, communication between threads is one of the most common topics. STACKOVERFLOW users want to send messages from one thread to another, for instance, to update the user interface (UI), which is running in the main thread (e.g., Q4369537). This is particularly true in mobile applications (e.g., Q4369537)

Concurrent libraries/frameworks In this theme we grouped questions that ask for indications about concurrent programming libraries or frameworks (15 questions asked). For instance:

Q3629784 ~ “How is Node.js inherently faster when it still relies on Threads internally?”

Q6916385 ~ “Is there a concurrent List in Java’s JDK?”

Q3847108 ~ “What is the Haskell response to Node.js?”

From this theme, most of the questions (5 of them) are about the Node.js framework (e.g., Q3629784, Q4631774, Q3011317). However, we did not find mention to concurrency-related frameworks, such as Java’s ForkJoin framework [13] or software transactional memory.

Correctness Finally, in this theme we group questions related to the correctness of programs when employing concurrent programming constructs (10 questions asked). For instance:

Q9666 ~ “Is accessing a variable in C# an atomic opera-

⁶ <http://docs.oracle.com/javase/1.5.0/docs/guide/misc/threadPrimitiveDeprecation.html>

tion?”

Q7095 ~ “Is the C# constructor thread safe?”

Q680097 ~ “I’ve heard `i++` isn’t thread safe, is `++i` thread-safe?”

It is well-known that correctness is one of the biggest problems associated with concurrent programming. STACKOVERFLOW users also discuss this topic in great detail. However, such questions are not always straightforward to answer. For instance, when a STACKOVERFLOW user questioned “Is the `!=` check thread safe?” (Q18460580), the STACKOVERFLOW user who wrote the accepted answer needed to analyze the byte code generated by the `!=` instruction. She observed that the variables under test needed to be loaded twice, which makes it a non-atomic operation.

Let us now investigate different quantitative aspects of each one of these themes. Table 2 shows the results. We use **boldface** to highlight the highest value for each case.

From this table we can observe that the majority of questions (153 of them) are related to the basics of concurrent programming. We consider a question as basic if it falls in one of the following themes: Theoretical Concepts, Practical Concepts, First Steps or Thread Life-Cycle. We believe that these questions are more popular because programmers are still getting acquainted with concurrency. Also, since the concurrent programming learning curve is not negligible, such questions are supposed to be revisited (the \mathbb{V} variable) and favorable (the \mathbb{F} variable) more often. This is particularly true for the Thread Life-Cycle theme (\mathbb{V} is 65.56), since it has a couple of different stages.

Second, the A/Q ratio is 8.5x greater than the mean ratio on STACKOVERFLOW. When a question has much more answers than the average, we believe it is because the question is *easier to answer*. This is particularly true for the Theoretical Concepts (A/Q is 10.56, \mathbb{A} is 6.14). For instance, answers to questions in this theme can easily be found in well-known concurrent programming textbooks (e.g., [6, 12]). In our base group, no question has received less than 2 answers.

Third, the STACKOVERFLOW user who asks a question can mark at most one answer per question as accepted. A STACKOVERFLOW user earns additional points when her answer is marked as accepted. When a question receives more attention, for instance, in terms of answers and visualizations, the user who came up with the accepted answer can receive additional points, because other STACKOVERFLOW users can “upvote” that answer, meaning that this answer was *useful* for other users too. We use this feature to define the success of a question, represented at the “Solved” column. As we can see, the majority of questions from our base group are successfully answered (89.45% of them — 39.99% on STACKOVERFLOW).

3.2 RQ2. Concepts developers want to know more

We now provide a deeper discussion based on the Theoretical and Practical Concepts themes. Analyzing these themes, we found 4 and 3 new sub-themes, respectively for Theoreti-

cal Concepts and Practical Concepts themes. We discuss the two with the highest number of questions next.

3.2.1 Theoretical Concepts

Threading Threading is the most common sub-theme with 25 concurrent programming-related questions. These questions vary from the difference between “*implements Runnable and extend Threads*” (Q541487), or “*the difference between a process and a thread*” (Q200469). Variations of these questions can be found in this sub-theme (e.g., Q4130194, Q1762418 and Q807506), and most of them are related to well-established and studied concurrent programming constructs. The only exception is a question that asks about the `std::promise` concurrent construct, introduced in the C++11 standard library (Q11004273). However, the concept *promise* was proposed decades ago [5].

Locks We found 16 questions regarding locks. They vary from very basic questions, such as “*what is a mutex?*” (Q34524), to more elaborated ones, such as the difference between “*Recursive Lock (Mutex) vs Non-Recursive Lock (Mutex)*” (Q187761). Most surprisingly is the recurring interest on semaphores (10 questions about it). In a recent study among more than 2,000 Java projects, the authors observed that none of them employs the Semaphore class [23].

3.2.2 Practical Concepts

Threading Threading is also the most common sub-theme with 23 questions. In this sub-theme, developers are interested in understanding “*When to use a thread pool in C#?*” (Q145304), or if the `time.sleep()` construct puts a thread or a process to sleep (Q92928).

Locks We found 17 questions regarding locking strategies. One example of such question is “*What is a good pattern for using a Global Mutex in C#?*” (Q229565). In this question, in particular, the authors stressed the fact that the concept is *misunderstood*, and ask for advice in the following scenarios: (1) while guaranteeing that the mutex is properly released; (2) deals with cases where other processes abandon the mutex. We also observed that this topic has questions that we consider simple ones, for instance, “*How to use wait and notify in Java?*” (Q886722), and questions that we consider hard ones, such as “*When should one use a spinlock instead of mutex?*” (Q5869825). We consider the latter question as hard to answer, because answering it requires in-depth knowledge about locks implementations [2].

3.3 RQ3. Unanswered questions

Analyzing all the 28 questions that do not have an accepted answer, we observed that 22 of them can be described as “weakly-accepted”, that is, a question has an answer that one can judge as accepted, but the author of the question did not mark it as so. One example is question Q14010906, where the STACKOVERFLOW user discovered that a synchronization point was introduced in the HashMap class, causing a

Table 2. The distribution of questions and answers per theme. A/Q means the proportionality of answers per question.

Themes	Questions	Answers	A/Q	S	A	C	F	V	P	Solved
Theoretical Concepts	55	581	10.56	87.78	6.14	1.39	31.03	47.46	16.16	83.63%
Practical Concepts	51	413	8.09	63.73	4.71	1.17	21.05	36.31	12.18	88.23%
First Steps	29	251	8.65	56.4	5.03	1.11	18.48	60.96	12.88	79.31%
Thread Life-Cycle	18	167	9.27	59.58	5.39	1.72	17.37	65.56	14.44	94.44%
Technical problems	53	433	8.16	51.65	4.75	1.34	19.26	39.22	11.99	90.56%
Correctness	10	91	9.1	49.31	5.29	1.85	11.99	15.55	9.79	90%
Concurrency libs/frameworks	15	87	5.8	44.63	3.37	2.94	16.98	23.19	11.17	100%

great impact on the performance on clients, as a STACKOVERFLOW user mentioned, “*when I run on 64 threads, I get less performance than when I run on 1 thread*”. The author of this modification acknowledged the bottleneck and came up with a fix for the problem. The author of the question did not accept the answer, though.

Analyzing the remaining 6 unanswered questions revealed interesting research problems. First, question Q12159 asked for advice on testing multi-threaded code. The author stressed the fact that this is a “*key problem for programmers today*”. This particular question has 26 answers. However, even the answer with the highest score is vague. For instance, the user suggests programmers to “*reduce the complexity of threaded code as much as possible*” or to “*identify those places in your design where threads interact with the same instance and reduce the number of those places*”. Such suggestions, albeit helpful to reduce code complexity, they do not give any insight on how to test multi-threaded code. Second, two STACKOVERFLOW users have mentioned the “Humble Object” testing pattern. However, this pattern does not prevent the occurrence of concurrency bugs. Third, STACKOVERFLOW users suggested the use of static analysis, dynamic analysis and model checkers tools, such as FindBugs and Java Pathfinder. Prior research also pointed out that FindBugs is useful in reporting incorrect usage of synchronization and wait and notify mechanisms [9].

Another interesting question is related to the design and implementation of a lock-free circular buffer (Q871234). Several users pointed out that this is still an open research problem, and if the asker “*have not spent at least six months studying lock-free data structures, do not attempt to write one yourself*”. Another questions is regarding how to programmatically identify deadlocks in Java (Q217113). This question is quite polemic, since some users have pointed out that it is not possible, whereas some users pointed out that yes, it is possible to detect deadlocks using the ThreadMXBean class, introduced in JDK 1.5. However, as pointed out by a respondent in the same question, the ThreadMXBean class only work for synchronized blocks, and not the new java.util.concurrent mechanisms.

4. Further Analysis

First, although GPUs are easily available in the market and are heavily used by the gaming industry, we only found one question regarding GPU programming (Q2392250). We decided to further investigate this issue by including the “CUDA”, “OpenCL”, “GPU” and “GPGPU” keywords in our search string. However, no new question about it was found. Still, we found two questions about the actors programming model (Q1251666, Q2708033), but newer concurrent constructs such as the ForkJoin framework, Java parallel streams, or software transactional memory were never mentioned. This fact is understandable, since it takes time for a new feature to become popular.

Second, we observed that concurrent programming has reached mobile developers. We found 15 questions regarding thread management in Android (e.g., Q4369537), 6 questions on iOS (e.g., Q7055424) and 7 questions on Windows Phone (e.g., Q4331262). Most of these questions are related to the default concurrent programming constructs available on the programming languages. However, we also observed that STACKOVERFLOW users are employing the specific threading constructs available only for these platforms, such as the AsyncTask concurrent construct for the Android platform (e.g., Q9654148), or the NSOperation concurrent construct for the iOS platform (e.g., Q830218).

Third, we also observed another brick on the wall of the programming language wars [25]. For instance, a STACKOVERFLOW user asked “*Is there a language that makes creation of parallel programs as easy as object-oriented programming languages help creating complex architectures?*” (Q134867). Answers to this question came in different flavors: Erlang (4 answers), Clojure (3 answers), and Go (3 answers). Occam, F#, Java, Mozart Oz, Ciao and Scala were supported by one STACKOVERFLOW user each. This fact shows that the “One Language to Rule Them All” hypothesis [25, p. 3] does not hold also for concurrent programs. Interestingly, however, most of the users have cited functional languages. Likewise, it has been argued for many years that functional programs are well-suited for parallel evaluation [8]. STACKOVERFLOW users observed these benefits, for instance, when asked “*How/why do functional*

languages (specifically Erlang) scale well?” (Q474497), a STACKOVERFLOW user answered that “A functional language doesn’t (in general) rely on mutating a variable. This in turn avoids the majority of the hoop jumping that traditional languages have to go through to implement an algorithm across processors or machines.”.

Fourth, STACKOVERFLOW users seem to like to learn from mistakes. We observed some questions where a user asked for a real world example, such as deadlocks (e.g., Q8880286) and livelocks (e.g., Q1036364). We believe that these questions are important because sometimes the concept is easy to get, however, in practice, without seeing a real example of the problem, the problem might be hidden during a code review process.

Fifth, we did not find questions that ask for advices on how to use concurrent programming constructs to improve application performance, which is surprising, since performance is one of the most important motivations for the use of concurrency and parallelism [15]. Also, we believe that this is an important topic that deserves more discussion because, when misused, concurrent programming techniques can introduce performance bugs [17].

Sixth, in our base group, none of questions were created in 2014. We further analyzed this matter by selecting the top-10 most popular questions created in 2014. We observed that all these questions have answers, although four of them do not have an accepted answer. As regarding the popularity, we found that the most popular question in 2014 (Q21163108) has \mathbb{P} equals to 82.35 (11.5x lesser than the highest popularity of our base group).

Finally, we observed that Java and C# are the programming languages which have more questions about (77 and 60, respectively), which is not surprising, since these two languages have well-known and well-studied concurrent programming constructs. Conversely, we also observed that non-traditional languages for concurrent programming, such as Python (17 questions), are of great interest to the STACKOVERFLOW community. Even though recent releases of the language have introduced advanced high-level concurrent programming constructs, such as `concurrent.futures` (released in v3.2) and `asyncio` (release in v3.4), all of the questions pertain to the basic threading constructs (e.g., Q31340, Q92928, Q3033952).

5. Implications

First, we observed that several questions are related to the basics of concurrent programming concepts and usage. Thus, both researchers and educators can get these results to improve the knowledge in this field. Researchers can come up with new empirical studies on the ease to use of such concurrent programming constructs, whereas educators can provide additional homework, exams, and class activities, focusing on real word problems provided in this study. Tools to help developers to write concurrent software from

the very beginning are also needed. Such tools can be in the forms of e-learning environments, recommendation systems, and static analysis ones to identify concurrent-related problems. Furthermore, we observed the need of a better API documentation that also provides the simplest working code example. This is important for at least two reasons: (1) several STACKOVERFLOW users have mentioned that they found the documentation is *misleading* (e.g., Q6964011); and (2) software developers often ask for the simplest working code (e.g., Q266168). API designers can improve the documentation of their libraries, explaining the most important differences between their libraries and other ones that are in widespread use.

6. Related Work

About empirical studies on concurrent programming (e.g., [14, 17, 18, 23, 24]), Pinto *et al.* [23] conducted a research over 2,000 projects, focusing on the Java concurrent constructs. Okur *et al.* [17] analyzed over 1,300 Windows Phone apps, and observed that developers are (1) missing opportunities to use a high-level asynchronous programming framework and they are (2) misusing the available constructs, creating problems that might hurt performance and introduce deadlocks. Rossbach *et al.* [24] conducted a study asking undergraduate students to resolve concurrent programming tasks using coarse- and fine-grained locks, monitors, and transactional memory to understand which synchronization technique is easier to use. Lu *et al.* [14] focused on concurrency bugs. The authors found that 30% of all concurrent bug are related to deadlocks. In contrast, in our study, in which we open to any kind of concurrency-related question, we found that less than 5% of the our questions are related to deadlocks.

About empirical studies on STACKOVERFLOW (e.g., [1, 16, 20, 21]), Barua *et al.* [1] proposed a semi-automatic approach to discover the main topics present in STACKOVERFLOW discussions. Morrison *et al.* [16] who studied how age is related to programming skills, and results show that programming knowledge increases until the age of 50. Previous effort has also been placed on understanding how developers are dealing with energy consumption issues [21], and what are the barriers to the adoption of refactoring tools [20].

However, to the best of our knowledge, there is no study in the literature focused on understanding what makes concurrent programming hard for software developers using a social platform as the data provider.

7. Threats to Validity

Internal factors. First, we mitigate the bias of choosing a non-representative set of questions by filtering out the most popular ones. Second, not all questions that we found are related to concurrent programming. We manually investigated all questions and answers to minimize the false-positive rate. Still, to reduce the false-negatives, we used the same keywords used by Lu *et al.* [14].

External factors. We mitigate representativeness of our subjects by mining STACKOVERFLOW, which is the most widely used forum in the software development world. However, our results only apply to application programmers interested in concurrent programming on STACKOVERFLOW. It does not cover application programmers that use other Q&A websites. Also, we are only considering English questions and answers. Although it is possible to collect data from other Q&A websites, we do not expect major divergences considering English content only.

8. Conclusions

We performed an empirical analysis of concurrency-related discussions on STACKOVERFLOW, a popular Q&A website. Our study involves analyzing the text of both questions and answers related to concurrent programming development to extract the dominant topics of discussion using a qualitative methodology. Our study provides important insights into better understanding what are the difficulties that application developers have when writing concurrent programs.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments. Gustavo is supported by CAPES/Brazil, Fernando is supported by CNPq/Brazil (304755/2014-1, 487549/2012-0 and 477139/2013-2), FACEPE/Brazil (APQ-0839-1.03/14) and INES (CNPq 573964/2008-4, FACEPE APQ-1037-1.03/08, and FACEPE APQ-0388-1.03/14).

References

- [1] A. Barua, S. Thomas, and A. Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. *EMSE*, 2012.
- [2] T. David, R. Guerraoui, and V. Trigonakis. Everything you always wanted to know about synchronization but were afraid to ask. In *SOSP*, 2013.
- [3] J. Fereday and E. Muir-Cochrane. Demonstrating rigor using thematic analysis: A hybrid approach of inductive and deductive coding and theme development. *International Journal of Qualitative*, 5, 2006.
- [4] P. Fleming and J. Wallace. How not to lie with statistics: The correct way to summarize benchmark results. *Commun. ACM*, 29(3):218–221, Mar. 1986. ISSN 0001-0782.
- [5] D. Friedman and D. Wise. *The Impact of Applicative Programming on Multiprocessing*. Technical report. Indiana University, Computer Science Department, 1976.
- [6] B. Goetz, T. Peierls, J. Bloch, J. Bowbeer, D. Holmes, and D. Lea. *Java Concurrency in Practice*. Addison-Wesley, 2006.
- [7] D. Huo, T. Ding, C. McMillan, and M. Gethers. An empirical study of the effects of expert knowledge on bug reports. In *ICSME*, 2014.
- [8] S. P. Jones. Parallel implementations of functional programming languages. *The Computer Journal*, 32(2):175–186, 1989.
- [9] D. Kester, M. Mwebesa, and J. Bradbury. How good is static analysis at finding concurrency bugs? In *SCAM*, pages 115–124, Sept 2010.
- [10] A. Kulkarni, Y. Liu, and S. Smith. Task types for pervasive atomicity. In *OOPSLA*, 2010.
- [11] T. LaToza and B. Myers. Hard-to-answer questions about code. In *PLATEAU*, pages 8:1–8:6, 2010.
- [12] D. Lea. *Concurrent Programming in Java. Design Principles and Patterns*. Addison-Wesley Longman Publishing., 2nd edition, 1999.
- [13] D. Lea. A java fork/join framework. In *Java Grande*, 2000.
- [14] S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from mistakes: A comprehensive study on real world concurrency bug characteristics. In *ASPLOS*, 2008.
- [15] P. McKenney. *Is Parallel Programming Hard, And, If So, What Can You Do About It?* kernel.org, Corvallis, OR, USA, 1st edition, 2014. URL <http://kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html>.
- [16] P. Morrison and E. Murphy-Hill. Is programming knowledge related to age? an exploration of stack overflow. In *MSR*, 2013.
- [17] S. Okur, D. Hartveld, D. Dig, and A. Deursen. A study and toolkit for asynchronous programming in c#. In *ICSE*, 2014.
- [18] V. Pankratius and A.-R. Adl-Tabatabai. Software engineering with transactional memory versus locks in practice. *Theory of Computing Systems*, 55(3):555–590, 2014. ISSN 1432-4350.
- [19] V. Pankratius, F. Schmidt, and G. Garretton. Combining functional and imperative programming for multicore software: An empirical study evaluating scala and java. In *ICSE*, 2012.
- [20] G. Pinto and F. Kamei. What programmers say about refactoring tools? an empirical investigation of stack overflow. In *WRT*, 2013.
- [21] G. Pinto, F. Castor, and Y. Liu. Mining questions about software energy consumption. In *MSR*, 2014.
- [22] G. Pinto, F. Castor, and Y. Liu. Understanding energy behaviors of thread management constructs. In *OOPSLA*, 2014.
- [23] G. Pinto, W. Torres, B. Fernandes, F. Castor, and R. S. Barros. A large-scale study on the usage of javas concurrent programming constructs. *Journal of Systems and Software*, 106(0):59–81, 2015. ISSN 0164-1212.
- [24] C. J. Rossbach, O. S. Hofmann, and E. Witchel. Is transactional programming actually easier? In *PPoPP*, 2010.
- [25] A. Stefik and S. Hanenberg. The programming language wars: Questions and responsibilities for the programming language community. In *Onward!*, pages 283–299, 2014.
- [26] H. Sutter and J. Larus. Software and the concurrency revolution. *Queue*, 3(7):54–62, Sept. 2005. ISSN 1542-7730.
- [27] J. von Neumann. First draft of a report on the edvac. *IEEE Ann. Hist. Comput.*, 15(4):27–75, Oct. 1993. ISSN 1058-6180.