What Do Concurrency Developers Ask About?
A Large-scale Study Using Stack Overflow

Syed Ahmed
Oakland University, USA
sfahmed@oakland.edu

Mehdi Bagherzadeh
Oakland University, USA
mbagherzadeh@oakland.edu

ABSTRACT

Background Software developers are increasingly required to write concurrent code. However, most developers find concurrent programming difficult. To better help developers, it is imperative to understand their interest and difficulties in terms of concurrency topics they encounter often when writing concurrent code.

Aims In this work, we conduct a large-scale study on the textual content of the entirety of Stack Overflow to understand the interests and difficulties of concurrency developers.

Method First, we develop a set of concurrency tags to extract concurrency questions that developers ask. Second, we use latent Dirichlet allocation (LDA) topic modeling and an open card sort to manually determine the topics of these questions. Third, we construct a topic hierarchy by repeated grouping of similar topics into categories and lower level categories into higher level categories. Fourth, we investigate the coincidence of our concurrency topics with findings of previous work. Fifth, we measure the popularity and difficulty of our concurrency topics and analyze their correlation. Finally, we discuss the implications of our findings.

Results A few findings of our study are the following. (1) Developers ask questions about a broad spectrum of concurrency topics ranging from multithreading to parallel computing, mobile concurrency to web concurrency and memory consistency to runtime speedup. (2) These questions can be grouped into a hierarchy with eight major categories: concurrency models, programming paradigms, correctness, debugging, basic concepts, persistence, performance and GUI. (3) Developers ask more about correctness of their concurrent programs than performance. (4) Concurrency questions about thread safety and database management systems are among the most popular and the most difficult, respectively. (5) Difficulty and popularity of concurrency topics are negatively correlated.

Conclusions The results of our study can not only help concurrency developers but also concurrency educators and researchers to better decide where to focus their efforts, by trading off one concurrency topic against another.

CCS CONCEPTS
• General and reference → Empirical studies; • Theory of computation → Concurrency;

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.

ESEM ’18, October 11–12, 2018, Oulu, Finland
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5823-1/18/10…$15.00
https://doi.org/10.1145/3239235.3239524

1 INTRODUCTION

Software developers are increasingly required to write concurrent code to satisfy both functional and nonfunctional requirements of their software. For example, software with a graphical user interface must be concurrent in order to satisfy a functional requirement that the software should be able to display more than one window at a time. Similarly, software with performance requirements must be concurrent in order to satisfy a nonfunctional requirement that the software should support better performance. Software is concurrent if its different computations can potentially run at the same time and otherwise is sequential. However, most developers think sequentially [19] and find concurrent programming difficult.

To better help developers with concurrent programming, it is imperative to understand their interests and difficulties in terms of the concurrency topics they encounter often when writing concurrent code and their difficulties when working with these topics. Such understanding not only can help concurrency developers but also education, development and research communities that support these developers to better decide when and where to focus their efforts [2, 5, 7, 23, 24, 28, 31, 34]. Without such understanding, developers may not prepare themselves for similar difficulties, educators may develop the wrong educational material and researchers may make incorrect assumptions.

With more than three million developer participants, thirty eight million question and answer posts in two billion words, Stack Overflow [30] has become a large and popular knowledge repository for developers to post questions, receive answers and learn about a broad range of topics. This makes Stack Overflow a great source to learn about developers’ interests and difficulties [5, 7, 24, 25, 28, 34].

To understand interests and difficulties of concurrency developers, we conduct a large-scale study on the textual content of the entirety of Stack Overflow to answer these research questions:

• RQ1. Concurrency topics What concurrency topics do developers ask questions about?

• RQ2. Topic hierarchy What categories do these concurrency topics belong to? What does the hierarchy of these concurrency topics look like?
RQ3. Popularity What topics are more popular among concurrency developers?

RQ4. Difficulty What topics are more difficult to successfully find answers to their questions?

RQ5. Correlation What popular concurrency topics are more difficult? How do popularity and difficulty of concurrency topics correlate?

To answer these questions, we take the following major steps. First, we develop a set of concurrency tags to identify and extract concurrency questions that developers ask on Stack Overflow. Second, we use latent Dirichlet allocation (LDA) [9] topic modeling and an open card sort [11] to manually determine the topics of these questions using their textual contents. Third, we construct a topic hierarchy by repeated grouping of similar topics into categories and lower level categories into higher level categories. Fourth, we investigate the coincidence of our concurrency topics with findings of previous work. Fifth, we measure the popularity and difficulty of our concurrency topics using several well-known metrics used by previous work [5, 7, 22, 28, 28, 31, 34] and analyze their correlation. Finally, we discuss the implications of our findings for concurrency developers, educators and researchers.

A few findings of our study are the following. Concurrency topics: (1) Developers ask questions about a broad spectrum of concurrency topics ranging from threading to parallel computing, mobile concurrency to web concurrency and memory consistency to runtime speedup. Topic hierarchy: (2) These questions can be grouped into a hierarchy with eight major categories: concurrency models, programming paradigms, correctness, debugging, basic concepts, persistence, performance and GUI. (3) Developers ask more about correctness of their concurrent programs than performance. (4) Our concurrency topics and categories coincide with several findings of previous work by Pinto et al. [25], Barua et al. [7], Rosen and Shihab [28] and Lu et al. [19], among others. Popularity & difficulty and their correlation: (5) Concurrency questions about thread safety and database management systems are among the most popular and the most difficult, respectively. (6) Difficulty and popularity of concurrency topics are negatively correlated.

Our dataset is available at https://goo.gl/uYCQPU.

2 METHODOLOGY

Figure 1 shows an overview of the methodology used to study interests and difficulties of concurrency developers on Stack Overflow.

Step 1: Download Stack Overflow dataset In the first step of our analysis, we download the Stack Overflow dataset which is publicly available through Stack Exchange Data Dump [29]. The dump includes a large set S of question and answer posts with a set of data for each post. Among others, the data for a post includes its identifier, its type (question or answer), title, body, tags, creation date, view count, score, favorite count and the identifier of the accepted answer for the post if the post is a question. An answer to a question is accepted if the contributor who posted the question marks it as accepted. Our dataset includes 38,485,046 questions and answers posted over a time span of over 9 years from August 2008 to partway through December 2017 by 3,589,412 developer participants of Stack Overflow. Among these posts 14,995,834 (39%) are questions and 23,489,212 (61%) are answers of which 8,034,235 (21%) are marked as accepted answers.

Step 2: Develop concurrency tag set We consider a post as a concurrency post if it has a concurrency tag. To develop a set of concurrency tags we take the following steps. First, we manually inspect Stack Overflow’s top 100 most used tags and select its concurrency-related tags to form the initial set \( T_0 \) of concurrency tags. \( T_0 \) includes a single tag “multithreading” [19, 25]. Second, we go through the Stack Overflow dataset \( S \) and extract questions \( P \) whose tags contain a tag from \( T_0 \). The set \( P \) includes 103,747 questions. Third, we extract tags of the posts in \( P \) to form the set of candidate concurrency tags \( T \). Finally, we use two heuristics \( \alpha \) and \( \beta \) to refine \( T \) by keeping tags that are significantly relevant to concurrency and excluding others. The heuristic \( \alpha \) measures the relevance of a tag \( t \) in \( T \) to concurrency.

\[
\alpha = \frac{\text{number of posts with tag } t \text{ in } P}{\text{number of posts with tag } t \text{ in } S}
\]

Similarly, the heuristic \( \beta \) measures the significance of a tag \( t \) in \( T \).

\[
\beta = \frac{\text{number of posts with tag } t \text{ in } P}{\text{number of posts in } P}
\]

We consider a tag \( t \) to be significantly relevant to concurrency if its \( \alpha \) and \( \beta \) values are higher than or equal to specific thresholds. Our experiments using a broad range of thresholds for \( \alpha \) and \( \beta \) show that \( \alpha = 0.1 \) and \( \beta = 0.01 \) allows for a significantly relevant set of concurrency tags. With these threshold values the set \( T \) of our concurrency tags becomes

\[
T = \{ \text{concurrency locking multiprocessing multithreading mutex parallel-processing pthreads python-multithreading synchronization task-parallel-library thread-safety threadpool} \}
\]

Note that after refinement, the concurrency tag set \( T \) includes tags like “concurrency”. Also, using \( T \) to identify concurrency posts does not prevent a concurrency post to also have tags, such as “asynchronous”, that are not explicitly in \( T \). Our threshold values are consistent in range with thresholds used by previous work.

To illustrate, our set \( C \) of concurrency posts, with tags \( T \), includes questions with “asynchronous” tag. In addition, topics such as task parallelism and mobile concurrency in Table 1 cover posts related to asynchrony.
What Do Concurrency Developers Ask About?  

Table 1: Concurrency tags for select relevance and significance threshold values. Our concurrency tag set \( T \) is in gray.

<table>
<thead>
<tr>
<th>((\alpha, \beta))</th>
<th>Set of tags</th>
<th>No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>((0.3, 0.015))</td>
<td>concurrency multithreading pthreads thread-safety threadpool</td>
<td>5</td>
</tr>
<tr>
<td>((0.3, 0.01))</td>
<td>concurrency multithreading mutex pthreads python-multithreading thread-safety threadpool</td>
<td>7</td>
</tr>
<tr>
<td>((0.3, 0.005))</td>
<td>backgroundworker concurrency executor-service multithreading mutex pthreads python-multithreading runnable semaphore synchronization thread-safety threadpool</td>
<td>12</td>
</tr>
<tr>
<td>((0.2, 0.015))</td>
<td>concurrency locking multithreading pthreads synchronization thread-safety threadpool</td>
<td>7</td>
</tr>
<tr>
<td>((0.2, 0.01))</td>
<td>concurrency locking multithreading mutex pthreads python-multithreading synchronization task-parallel-library thread-safety threadpool</td>
<td>10</td>
</tr>
<tr>
<td>((0.2, 0.005))</td>
<td>atomic backgroundworker concurrency deadlock executor-service grand-central-dispatch locking multithreading mutex openmp pthreads python-multithreading runnable semaphore synchronization task-parallel-library thread-safety threadpool wait</td>
<td>20</td>
</tr>
<tr>
<td>((0.1, 0.01))</td>
<td>concurrency locking multithreading parallel-processing pthreads synchronization thread-safety threadpool</td>
<td>8</td>
</tr>
<tr>
<td>((0.1, 0.005))</td>
<td>atomic backgroundworker concurrency deadlock executor-service grand-central-dispatch locking multithreading mutex openmp parallel-processing pthreads python-multithreading queue runnable semaphore synchronization task-parallel-library thread-safety threadpool wait</td>
<td>12</td>
</tr>
<tr>
<td>((0.05, 0.015))</td>
<td>asynchronous c++11 concurrency locking multithreading parallel-processing pthreads sockets synchronization thread-safety threadpool</td>
<td>11</td>
</tr>
<tr>
<td>((0.05, 0.01))</td>
<td>android-async-task asynchronous boost c++11 concurrency locking multithreading mutex parallel-processing pthreads python-multithreading sockets synchronization task-parallel-library thread-safety threadpool timer</td>
<td>18</td>
</tr>
<tr>
<td>((0.05, 0.005))</td>
<td>android-async-task async-await asynchronous atomic backgroundworker boost c++11 concurrency deadlock executor-service grand-central-dispatch locking multithreading mutex openmp parallel-processing process pthreads python-multithreading queue runnable semaphore sockets synchronization task-parallel-library thread-safety threadpool timer wait</td>
<td>32</td>
</tr>
</tbody>
</table>

[7, 34]. Rosen and Shihab [28] and Yang et al. [34] use similar approaches to develop their set of security and mobile tags. Table 1 shows the set of concurrency tags for a select set of threshold values for \( \alpha \) and \( \beta \) with \( T \) in gray.

**Step ❶: Extract concurrency posts** After developing the set of concurrency tags \( T \), we extract Stack Overflow posts whose tag set contains a tag in \( T \). This set includes 156,777 question and 249,662 answer posts of which 88,764 (36%) are accepted answers. To reduce noise, following previous work [7, 28], we add questions and their accepted answers from this set to the set of concurrency post \( C \) and discard unaccepted answers. \( C \) includes 156,777 questions and 88,764 accepted answers, i.e. 245,541 posts in total.

**Step ❷: Preprocess concurrency posts** We preprocess the set of concurrency posts \( C \) to reduce the noise [7, 34] for the next steps of the analysis, by taking the following actions. First, we remove code snippets, enclosed in \( \langle code \rangle \), HTML tags, such as \( \langle p \rangle \) and \( \langle /p \rangle \), stop words, such as “a”, “the” and “is” and numbers, punctuation marks, non alphabetical characters and URLs. Second, we reduce words to their base representations. For example “reading”, “read” and “reads” all reduce to their base “read”. For stop words and word reduction we use MALLET’s [27] list of stop words and Porter stemming algorithm [27], respectively.

**Step ❸: Infer and label topics** After preprocessing, we use latent Dirichlet allocation (LDA) [9] to automatically infer topics through an unsupervised topic modeling of textual contents of concurrency posts \( C \). In a topic model, a document is a probabilistic distribution of topics where the topic itself is a probabilistic distribution of words. A topic is a set of frequently co-occurring words that approximates a real-world concept. In our analysis, a document is a question or an answer post. A document can have multiple topics that cover various proportions of the document and a topic can span over multiple documents. To illustrate, the set of co-occurring words \{task, execute, async, complete, run, cancel, wait, asynchronous, parallel, schedule\} is a topic that approximates the “task parallelism” concept. Task parallelism is a concurrent programming model that allows partitioning a computation into tasks, assigning tasks to concurrently running threads/processes for execution and collecting tasks’ results with their completion.

Several implementations of LDA are available. We use MALLET [21] to train a topic model with \( K \) topics and \( I \) iterations using standard values \( 30/K \) and 0.01 for MALLET’s hyperparameters, following previous work [5, 7, 8, 28, 34]. Our experiments with a broad range of values show that \( K = 30 \) and \( I = 1,000 \) allows for the inference of sufficiently granular topics. To produce the model, we treat each individual question post and accepted answer post as an individual document. MALLET processes 245,541 documents.

Topic inference produces a set of words as topics and their proportions. However, the inference cannot automatically assign name labels to these topics. Following previous work [5, 7, 22, 28, 34], we use an open card sort [11] to assign a label to a topic word set \( w \) by manually inspecting the top 20 words in \( w \) and reading through 15 random posts with \( w \) as their dominant topic word set. A topic is the dominant topic of a document if the proportion of the document that the topic covers is higher than proportions that other topics of the document cover. In an open sort, the sorting begins with no predefined categories and participants develop their own categories. The two authors individually assigned topics to word sets, reiterated and refined topics as necessary, and then mutually agreed on a final set of topics. The second author is a Programming Languages professor with extensive expertise in concurrent and event-based systems and the first author is a graduate student with coursework in concurrent and distributed systems. During topic labeling, we merged two pairs of topics because they had sufficiently similar keywords and questions. Pairs are merged into object-oriented concurrency and basic concepts topics. We removed one unrelated topic that is about synchronization between local and remote repositories in version control systems such as Git and is not about concurrency. The initial level of inter-rater agreement,
Table 2: Names, categories (separated by `\`) and top 10 words (stemmed) for our concurrency topics of Stack Overflow.

<table>
<thead>
<tr>
<th>No.</th>
<th>Topic name</th>
<th>Category</th>
<th>Topic words</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic concepts</td>
<td>basic concepts</td>
<td>code question work answer understand read find edit issu make</td>
</tr>
<tr>
<td>2</td>
<td>task parallelism</td>
<td>concurrency models</td>
<td>task execut async complet run cancel wait asynchron parallel schedul</td>
</tr>
<tr>
<td>3</td>
<td>producer consumer</td>
<td>concurrency models</td>
<td>queue messag consum produc process item buffer block wait empti</td>
</tr>
<tr>
<td>4</td>
<td>parallel computing</td>
<td>concurrency models</td>
<td>parallel node loop comput calcul openmp result algorithm mpi function</td>
</tr>
<tr>
<td>5</td>
<td>process life cycle</td>
<td>multiprocessing\ concurrency</td>
<td>models process child parent termin exit fork creat kill share start</td>
</tr>
<tr>
<td>6</td>
<td>python multiprocessing</td>
<td>multiprocessing\ concurrency</td>
<td>models python script run multiprocess process function modul command parallel php</td>
</tr>
<tr>
<td>7</td>
<td>thread life cycle</td>
<td>multithreading\ concurrency</td>
<td>models thread main creat run start execut background separ join finish</td>
</tr>
<tr>
<td>8</td>
<td>thread sharing</td>
<td>multithreading\ concurrency</td>
<td>models function variabl pass call pointer pthread argument type return global</td>
</tr>
<tr>
<td>9</td>
<td>thread scheduling</td>
<td>multithreading\ concurrency</td>
<td>models loop time wait stop run start sleep set check finish</td>
</tr>
<tr>
<td>10</td>
<td>thread pool</td>
<td>multithreading\ concurrency</td>
<td>models worker pool job number work process task creat limit threadpool</td>
</tr>
<tr>
<td>11</td>
<td>concurrent collections</td>
<td>correctness</td>
<td>list arrai map element collect iter number item kei add</td>
</tr>
<tr>
<td>12</td>
<td>thread safety</td>
<td>correctness</td>
<td>thread java safe multipl multi multithread concurr singl implement applic</td>
</tr>
<tr>
<td>13</td>
<td>locking</td>
<td>correctness</td>
<td>lock mutex wait condit releas semaphor acquird deadlock synchron resourc</td>
</tr>
<tr>
<td>14</td>
<td>memory consistency</td>
<td>correctness</td>
<td>read memori write variabl oper atom each share synchron access</td>
</tr>
<tr>
<td>15</td>
<td>entity management</td>
<td>persistence</td>
<td>concurrency session spring entiti transact actor collect updat model version</td>
</tr>
<tr>
<td>16</td>
<td>database management systems</td>
<td>persistence</td>
<td>databas tabl queri updat row record lock insert sql transact</td>
</tr>
<tr>
<td>17</td>
<td>file management</td>
<td>persistence</td>
<td>file read write log line open stream directori folder text</td>
</tr>
<tr>
<td>18</td>
<td>object-oriented concurrency</td>
<td>programming paradigms</td>
<td>object class method instanc creat static access refer variabl synchron</td>
</tr>
<tr>
<td>19</td>
<td>web concurrency</td>
<td>programming paradigms</td>
<td>request servic web server applic user app net respons http</td>
</tr>
<tr>
<td>20</td>
<td>event-based concurrency</td>
<td>programming paradigms</td>
<td>event signal handler timer callback call handl fire receiv slot</td>
</tr>
<tr>
<td>21</td>
<td>mobile concurrency</td>
<td>programming paradigms</td>
<td>android imag app activ game view updat frame asynctask devic</td>
</tr>
<tr>
<td>22</td>
<td>client-server concurrency</td>
<td>programming paradigms</td>
<td>server client connect send socket messag receiv data port read</td>
</tr>
<tr>
<td>23</td>
<td>data scraping</td>
<td>performance</td>
<td>data time problem load download work page structur solut url</td>
</tr>
<tr>
<td>24</td>
<td>runtime speedup</td>
<td>performance</td>
<td>time core cpu run perform memori number system process machin</td>
</tr>
<tr>
<td>25</td>
<td>unexpected output</td>
<td>debugging</td>
<td>code program work run output problem print line result function</td>
</tr>
<tr>
<td>26</td>
<td>irreproducible behavior</td>
<td>debugging</td>
<td>error test code run problem applic issu window compil crash</td>
</tr>
<tr>
<td>27</td>
<td>GUI</td>
<td>GUI</td>
<td>updat form gui window applic button control user progress click</td>
</tr>
</tbody>
</table>

before reiterations and refinements, using Cohen’s kappa score was 0.659 (moderate agreement).

Table 2 shows the inferred top 10 words that describe a topic, in the column topic words, and its manually assigned topic name, in the column topic name, for each topic of our 27 concurrency topics. This table shows stemmed topic words that are reduced to their base using the Porter stemming algorithm [27].

Step 9: Construct topic hierarchy We construct the topic hierarchy by repeated grouping of similar topics into categories and lower level categories into higher level categories.
Table 2 and Figure 3 show the textual and pictorial representations of the topic hierarchy. To illustrate, thread life cycle management and thread scheduling topics are grouped into a lower level category called multithreading where multithreading itself is grouped into a higher level category called concurrency models. The higher level category concurrency models includes other categories such as multiprocessing and parallel computing.

**Step \(\Theta\): Determine topic popularity** We measure the popularity of a concurrency topic using three metrics, used by previous work. The first metric is the average number of views for questions with the topic as their dominant topic [5, 22, 28, 34]. This metric includes views by both registered users and visitors of Stack Overflow. The inclusion of visitors’ views is important because in Stack Overflow there are many more visitors than there are registered users [20]. The second metric is the average number of questions of the topic marked as favorite by users [5, 22, 25, 34]. The third metric is the average score of questions of the topic [5, 22, 25, 34]. Intuitively, a topic with higher number of views and favorites and a higher score is more popular.

Table 3 shows popularity measurements of concurrency topics.

**Step \(\Theta\): Determine topic difficulty** We measure the difficulty of a concurrency topic using two metrics, used by previous work. The first metric is the number of questions with the topic as their dominant topic [28, 34]. And the second metric is the average median time needed for a question to receive an accepted answer [28, 34]. Intuitively, a topic with higher chance of its questions not receiving accepted answers or taking longer to receive accepted answers is more difficult.

Table 4 shows difficulty measurements of concurrency topics.

**Step \(\Theta\): Determine correlations** After determining popularity and difficulty, we use Kendall correlation tests to identify correlations, if any, between the three popularity metrics and two difficulty metrics of our concurrency topics.

In this paper, the popularity and difficulty of our concurrency topics are bound by their corresponding metrics defined in steps 7 and 8, which are different from other notions of difficulty [15].

### 3 RESULTS

In this section, we present and discuss the results of our study for research questions RQ1–RQ5. We also investigate the coincidence of our results with findings of relevant previous works.

#### 3.1 RQ1: Concurrency Topics

Topics of concurrency questions that developers ask on Stack Overflow are determined using LDA topic inference and topic labeling, as discussed in Section 2. Table 2 shows these concurrency topics.

As Table 2 shows, developers ask questions about a broad spectrum of concurrency topics ranging from thread pool to parallel computing, mobile concurrency to web concurrency and memory consistency to runtime speedup.

The meaning of these concurrency topics may be best understood by looking at questions that developers ask about in each of these topics. To illustrate, the following is a question in the thread pool topic in which the developer is asking how to implement a thread pool where the size of the thread pool can change based on its number of jobs. The Stack Overflow identifier for this question is 11249342 and it can be accessed at https://stackoverflow.com/questions/11249342.

**Q. 11249342** Creating a dynamic (growing/shrinking) thread pool I need to implement a thread pool in Java (java.util.concurrent) whose number of threads is at some minimum value when idle, grows up to an upper bound (but never further) when jobs are submitted into it faster than they finish executing, and shrinks back to the lower bound when all jobs are done … How would you implement something like that?…

Similarly, the following is a question in the web concurrency topic in which the developer is asking how to send emails using background threads in their web application where the background thread prevents blocking of the main thread and therefore does not force the user to wait until the email is sent. Classic ASP is a scripting language to write server side web applications.

**Q. 17052243** How to perform multithreading/background process in classic ASP I need to send emails via a background job on a classic-ASP app so the user doesn’t have to wait for a slow webserver to complete sending the email. I know I can use Javascript to generate two separate requests, but I’d rather not require Javascript. Plus, I suspect there’s a better way to pull this off. Ideas?

Finally, the following is a question in basic concepts topic that asks about basic motivations behind the need for concurrency.

**Q. 541344** What challenges promote the use of parallel/concurrent architectures? However, I am so used to thinking about solutions in a linear/serial/OOP/functional way that I am struggling to cast any of my domain problems in a way that merits using concurrency.

Before proceeding to the next research question, we examine the number of questions that developers ask for concurrency topics. Figure 2 shows the number of questions for these topics and their percentages. As Figure 2 shows, the numbers of questions that developers ask in different concurrency topics are not uniform. Developers ask the most number of questions (8%) about basic concepts which is inline with the general understanding that concurrency remains difficult for developers and they still ask questions about its basics. Developers ask the least number of questions (1%) about event-based concurrency.

Pinto et al. [25] study the 250 most popular concurrency questions on Stack Overflow. Our observation that the concurrency
In addition, the number of questions that developers ask in each category is not uniform. Developers ask the most questions (28%) about the concurrency models category and the least (5%) about GUI. In addition, developers ask more questions (12%) about correctness of their concurrent programs than their performance (7%). This is inline with the general tradeoff between performance advantages of concurrency and its correctness issues [3, 4, 16, 18].

Finding 3: Questions that developers ask about concurrency can be grouped into eight major categories: concurrency models, programming paradigms, correctness debugging, basic concepts, persistence, performance and GUI.

Finding 4: Developers ask the most (28%) about concurrency models and the least (5%) about GUI.

Finding 5: Developers ask more (12%) about concurrency correctness than performance (7%), inline with the tradeoff between concurrency’s performance benefits and correctness issues.

We continue this section by a detailed discussion of concurrency categories and their constituent topics and their coincidence with relevant previous works.

3.2 RQ2: Topic Hierarchy

A hierarchy for concurrency topics that developers ask questions about on Stack Overflow is constructed by repeated grouping of similar topics into categories and lower level categories into higher level categories, as described previously.

Figure 3 shows the hierarchy of concurrency topics with concurrency topics in gray and concurrency categories in white. The inner levels of the hierarchy are its higher levels and the hierarchy expands outwards to lower level categories and concurrency topics at the outermost level. Figure 3 also shows percentages for number of questions a lower level topic/category contributes to its higher level category.

As Figure 3 shows, the questions that developers ask about concurrency can be grouped into a hierarchy with eight high level categories: concurrency models, programming paradigms, correctness, debugging, basic concepts, persistence, performance and GUI.

Finding 1: Developers ask questions about a broad spectrum of concurrency topics ranging from thread pool to parallel computing, mobile concurrency to web concurrency and memory consistency to runtime speedup.

Finding 2: Developers ask the most (8%) about basic concepts, inline with the general understanding that concurrency remains difficult for developers and they still ask questions about its basics.

Finding 6: Developers ask the most (54%) about multithreading when asking about concurrency models, inline with the general understanding that multithreading is the de facto dominant concurrency model. Developers ask the least (6%) about producer consumer concurrency when asking questions about concurrency models.

In multithreading, developers ask questions with titles like (thread life cycle management): “How to safely destruct Posix thread pool in C++?” and (thread pool): “Is there a way to create a pool of pools using the Python workerpool module?”. Whereas producer consumer concurrency includes question with titles like “producer Consumer with Blocking Queues in Java EE as background task”. The name inside parentheses is the concurrency topic of the question.

Pinto et al. [25] categorize the 250 most popular concurrency questions into several concurrency themes. Our multithreading category and its thread life cycle management topic coincide with their threading and thread life cycle themes. Rosen and Shihab [28] study and categorize Stack Overflow questions related to mobile development into several mobile topics including threading. Our multithreading category coincides with their mobile threading topic.
3.2.2 Programming paradigms. Programming paradigms are mostly concerned about programming abstractions (e.g. objects and events), platforms (e.g. web and mobile) and patterns (e.g. producer consumer). The programming paradigms category includes five lower level categories among which object-oriented concurrency alone contains more than a third of questions that developers ask in this category. This is inline with the general understanding that object-orientation is a dominant programming paradigm. Developers ask the most (35%) about object-oriented concurrency and the least (6%) about event-based concurrency when asking about programming paradigms.

In the object-oriented concurrency and event-based concurrency topics developers ask questions with titles like “Is it better to synchronize object from inside of the class that encapsulates access it or from outside?” and “What type of timer event should I use for a background process when my timer fires very quickly?”.

Barua et al. [7] study and categorize all Stack Overflow questions and answers into several general topics including web and mobile development. Our concurrency topics mobile concurrency and web concurrency coincide with their general web development and mobile development topics. Our mobile concurrency topic is inline with Pinto et al.’s [25] observation that “concurrent programming has reached mobile developers”. Out of the 250 most popular concurrency questions in their study, 22 are related to mobile development.

Finding 7: Developers ask the most (35%) about object-oriented concurrency when asking about concurrent programming paradigms, inline with the general understanding that object-orientation is a dominant programming paradigm for concurrency. Developers, ask the least (6%) about event-based concurrency.

3.2.3 Correctness. Correctness is concerned with prevention of data corruption for concurrently accessed (e.g. read and write) data using mechanisms like locking, consistent memory models and thread safe data structures and programming patterns. Correctness questions are almost evenly divided among its topics thread safety, locking, concurrent collections and memory consistency.

In the correctness category developers ask questions with titles like (thread safety): “How to make factory [pattern] thread safe?” (locking): “Is there a way to lock 2 or more locks or monitors atomically?”, (concurrent collections): “Threadsafe dictionary that does lookups with minimal locking” and (memory consistency): “Atomic read-modify-write in C#”.

Lu et al. [19] categorize concurrency bug patterns and their fixes. Our memory consistency topic coincides with their observations that most concurrency bugs are atomicity violation bugs where the ‘desired serializability among multiple memory accesses is violated’ and order violation bugs where the ‘desired order between two (groups of) memory accesses is flipped.’ Our locking topic coincides with their designation that locking is one of the main fixes for concurrency bugs to ensure correctness. In addition, our correctness category and its locking topic coincide with Pinto et al.’s [25] correctness and locking themes. Similarly, our concurrent collections topic coincides with their concurrent libraries theme.

3.2.4 Basic concepts. Basic concepts include questions about both theoretical and practical questions about concurrency with titles like “How many threads are involved in deadlock?”, “What is a race condition?”, “Lock, mutex, semaphore... what’s the difference?” and “Java: notify() vs. notifyAll() all over again”. Our basic concepts category coincides with Pinto et al.’s [25] themes for theoretical concepts and practical concepts.

3.2.5 Debugging. Debugging is mainly concerned about finding and fixing concurrency bugs which manifest either in the behavior or output of programs. Debugging questions are almost evenly divided among its topics irreproducible behavior and unexpected output which includes question with titles like “Trace non-reproducible bug in C++” and “Synchronized codes with unexpected outputs”.

Our irreproducible behavior topic coincides with Lu et al.’s [19] observation that some “concurrency bugs are very difficult to repeat”.

3.2.6 Persistence. Persistence is about storing and retrieving of data using persistence management systems (e.g. databases management systems, entity/object persistence or file systems). Persistence includes three topics among which database management systems includes near half of persistence questions. The persistence category includes question with titles like (database management system): “How do I lock read/write to MySQL tables so that I can select and then insert without other programs reading/writing to the database?”, (file management): “Is this correct use of mutex to avoid concurrent modification to file?”, and (entity management): “Save entity using threads with JPA [Java Persistence API] (synchronized?”.


3.2.7 Performance. Performance is about speeding up execution of programs (e.g. data scraping programs). Performance includes two topics with question with titles like (runtime speedup): “Poor multithreading performance in .Net” and (data scraping): “Using multithreading to speed up web crawler written by beautifulsoup4 and python”.

Interestingly, Pinto et al. [25] mentions that they “did not find questions that ask for advice 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”. In contrast our performance topic includes more than 7% of all concurrency questions.

3.2.8 GUI. Graphical user interface allows for the interaction between a software and its user. GUI is the smallest category with regard to number of questions and includes question titles like “Force GUI update from UI Thread” and “Object synchronization with GUI Controls”.

Finding 8: Developers ask almost equally about thread safety (27%), locking (25%), concurrent collections (23%) and memory consistency (20%) when asking about correctness.

An entity management system automates serialization of objects (entities) for storage in database.

A data scraping program downloads data from remote web URLs and stores it locally.
Finding 9: Questions about thread safety are among the most popular whereas client-server concurrency questions are among the least popular.

3.3 RQ2: Popularity of Concurrency Topics

Popularity of concurrency topics is measured using average number of views of its questions, their average number of favorites and average scores, as described previously. Table 3 shows the popularity of concurrency topics using these metrics in a table, sorted by average number of views.

Intuitively, a topic with higher number of views and favorites and a higher score is more popular. In Table 3, the thread safety topic has the highest views, third highest favorites and second highest score whereas client-server concurrency has the third lowest views and lowest favorites and score.

Finding 10: Questions about database management systems are among the most difficult questions whereas memory consistency are among the easiest.

3.4 RQ4: Difficulty of Concurrency Topics

Difficulty of concurrency topics is measured using percentage of questions with no accepted answers and average median time to get an accepted answer, as described previously. Table 4 shows difficulty measurements using these metrics in a table, sorted by percentage of questions with no accepted answers.

Intuitively, a topic with higher percentage of its questions not receiving accepted answers or taking longer to receive accepted answers is more difficult. In Table 4, the irreproducible behavior topic has second highest percentage of questions with no accepted answers and highest time to accepted answers whereas memory consistency has the lowest percentage of questions with no accepted answers and second lowest time to accepted answers.

3.5 RQ5: Popularity/Difficulty Correlations

As discussed, thread safety is among the most popular concurrency topics but its difficulty is near bottom. Intuitively, this could suggest that there may be a correlation between the difficulty and popularity of concurrency topics. We confirm this intuition using Kendall correlation by taking the following steps. First, we calculate six correlations between each of our three popularity and two difficulty metrics. Second, for each correlation, we perform a significance test at the 90% confidence level to determine if the null hypothesis, of no significant correlation, can be rejected in favor of the alternative hypothesis, that there is a negative correlation between popularity and difficulty between all three popularity metrics and two difficulty metrics.

Interestingly, for all of six correlations, we find that there is sufficient evidence to conclude that its alternative hypothesis holds. For example, Pinto have calculated a negative correlation between popularity and difficulty metrics for all topics but its difficulty is near bottom. Intuitively, this could suggest that there may be a correlation between the difficulty and popularity of concurrency topics. We confirm this intuition using Kendall correlation by taking the following steps. First, we calculate six correlations between each of our three popularity and two difficulty metrics. Second, for each correlation, we perform a significance test at the 90% confidence level to determine if the null hypothesis, of no significant correlation, can be rejected in favor of the alternative hypothesis, that there is a negative correlation between popularity and difficulty between all three popularity metrics and two difficulty metrics.
What Do Concurrency Developers Ask About?

Table 5: Correlations of popularity and difficulty metrics.

<table>
<thead>
<tr>
<th>p-value</th>
<th>Avg. views</th>
<th>Avg. favorites</th>
<th>Avg. score</th>
</tr>
</thead>
<tbody>
<tr>
<td>% w/o acc. answer</td>
<td>0.0044</td>
<td>0.01166</td>
<td>0.001073</td>
</tr>
<tr>
<td>Hrs to acc. answer</td>
<td>0.001449</td>
<td>0.09196</td>
<td>0.02726</td>
</tr>
</tbody>
</table>

Figure 4: Trading off concurrency topics.

et al.’s [24] Measurement theme in their study of energy efficiency, is their most difficult and popular theme simultaneously. Similarly, the general mobile development topic that Barua et al. [7] finds popular is found to be difficult by Rosen and Shihab [28]. Note that the correlation between difficulty and popularity of concurrency topics does not imply causality. Wang et al. [32] study the relation of 46 factors with time to get an accepted answer to a question, which is one of our metrics for topic difficulty.

**Finding 11:** There is a statistically significant negative correlation between popularity and difficulty of concurrency topics.

4 IMPLICATIONS

The results of our study can not only help concurrency developers but also concurrency educators and researchers to better decide where to focus their efforts, by allowing them to trade off one concurrency topic against another based on their popularity and difficulty. To illustrate, Figure 4 shows the difficulty of our top 10 popular concurrency topics. For simplicity, popularity and difficulty equal the average number of views for questions of a topic and percentage of questions without accepted answers. In the figure, circles are topics with their size showing their number of questions.

**Developers** Using Figure 4, a novice concurrency developer may decide to focus their learning on task parallelism with higher popularity and less difficulty compared to database management systems. In contrast, a more knowledgeable developer who likes to learn about advanced topics with more than average difficulty may decide to learn about process life cycle management. Similarly, the manager of a development team can use Figure 4 to assign a less difficult task related to task parallelism to a more novice developer and a more difficult task related to database management systems to a more knowledgeable developer [34].

**Educators** Using Figure 4, an educator may decide to devote more material and teaching time to the more difficult thread pool topic compared to process life cycle management. Similarly, an educator can use Figure 4 to schedule teaching of the more popular and less difficult thread safety topic before thread scheduling.

**Researchers** Using Figure 4, a researcher may decide to focus their research project on the more difficult and slightly less popular thread pool rather than thread life cycle management in the hope of making contributions in a less crowded area.

Obviously, there are many factors that go into tradeoffs that developers, educators, and researchers make to decide where to focus their efforts. However, we believe our findings can contribute to inform and improve these decision making processes.

5 THREATS TO VALIDITY

In this section, we discuss threats to the validity of our study [33].

**Internal threats** Use of concurrency tags to identify concurrency posts is an internal threat to validity. This is because concurrency tags may not be able to identify the complete set of concurrency-related posts. To minimize this threat we use well-known techniques used by previous work [28, 34] in developing our concurrency tags and solid experiments with a broad range of tag relevance and significance thresholds α and β. Parsing Stack Overflow dataset, inferring topics from textual contents of posts and reduction of words to their bases is another threat. To minimize this threat, we use well known tools used by previous work. We parse Stack Overflow posts using Python elementTree XML API [28], infer topics using MALLER [5, 7, 28] and reduce words using the Porter stemming algorithm [5].

**External threats** Use of Stack Overflow as the only dataset to study interests and difficulties of concurrency developers is an external threat. This is because Stack Overflow posts may not be a representative of developer interests and difficulties. However, Stack Overflow’s large number of participant developers and posts along with its wide-spread popularity among developers may mitigate this risk. Also, we use title and body of not questions only but also their accepted answers to mitigate this risk.

**Construct threats** Manual labeling of topic word sets is a construct threat. To minimize this threat, we use a well-known approach used by previous work [5] to label topics using their top 10 words and 15 random questions. Determining an optimal value for K when inferring topics is another threat. To minimize this threat, we use a well-known approach used by previous work [5, 7, 28] to find a reasonable value for K using experiments with a broad range of values for K. It is well-known that determining an optimal value for K is difficult [7]. Heuristics to measure popularity and difficulty could be another threat. To minimize this threat, we use well-known heuristics and tools used by previous work to measure popularity [5, 28, 34] and difficulty [28, 31, 34].

6 RELATED WORK

Previous works that are closer to our work study software knowledge repositories, such as Stack Overflow, to understand interests [1, 2, 7, 13, 14, 28, 31] and difficulties [5, 28, 31, 34] of developers with software development topics.

**Concurrency** Closest to our work is the work of Pinto et al. [25] that uses the 250 most popular concurrency questions on Stack Overflow to study difficulties that developers face when writing concurrent programs. They categorize these difficulties into a set of themes including theoretical and practical concepts, threading and first steps themes, some of which coincide with our concurrency topics and categories.
In other previous work, Pinto et al. [26] analyze the code for 2227 projects to understand the usage of Java’s concurrent programming constructs and libraries and the evolution of the usage. Lin and Dig [17] study a corpus of 611 widely used Android apps to understand how developers use Android constructs for asynchronous concurrency. Blom et al. [10] study the usage of java.util.concurrent library in Qualitas corpus. Godefroid and Nagappan [12] survey 684 developers to study the spread and popularity of concurrency platforms and models at Microsoft.

In contrast, in this work we develop a set of concurrency tags to extract and study a large set of 245,541 concurrency posts on Stack Overflow and use latent Dirichlet allocation (LDA) to infer concurrency topics using textual contents of these posts and organize them into a topic hierarchy. In addition, we measure popularity and difficulty of concurrency topics, study their correlations and discuss their implications.

**Non-concurrency** Rosen and Shihab [28] use LDA to infer mobile development topics on Stack Overflow. They study popularity and difficulty of their mobile topics and categorize developers’ questions based on platforms for mobile development and type of questions that developers ask (why, what and how). Yang et al. [34] use LDA tuned with a genetic algorithm to infer security topics on Stack Overflow and manually organize their topics into five categories. They study popularity and difficulty of their security topics. Bajaj et al. [5] use LDA to infer client-side web development topics using Stack Overflow and study interests of developers in these topics and challenges they face when working with these topics. Barua et al. [7] use LDA to infer general topics on Stack Overflow. They study relations of questions and answers of these topics and evolution of developers’ interests in these topics both in general and for specific technologies.

Gyöngyi et al. [13] and Adamic et al. [1] study Yahoo!Answers posts to determine developer’s interests in a set of predefined categories. Hindle et al. [14] use LDA to infer topics related to development tasks from commit messages of a standalone software project and study evolution of developers’ interest in these topics. Treude et al. [31] and Allamanis and Sutton [2] study Stack Overflow posts to infer types of questions that developers ask and determine their difficulties with these question types. Bajacharya and Lopes [6] study logs of Koders, a code search engine, to learn about general topic of interest in code search.

However in this work, we focus on inferring concurrency topics in Stack Overflow using LDA, organize these concurrency topics into a hierarchy, study their popularity, difficulty and their correlations and discuss their implications.

7 **CONCLUSION AND FUTURE WORK**

In this paper, we performed a large-scale study using the textual content of the entirety of Stack Overflow to better understand interests and difficulties of concurrency developers. We inferred topics of concurrency questions that developers ask about, organized them into a topic hierarchy and measured their popularity and difficulty. We showed how our findings not only can help concurrency developers but also education, research and development communities that support these developers. One avenue of future work is to perform a similar study using commit logs and bug reports of publicly available concurrent software.

**REFERENCES**


[34] Xin-Li Yang, David Lo, Xin Xia, Zhu-Yuan Wan, and Jian-Ling Sun. What security questions do developers ask? a large-scale study of Stack Overflow posts. JCT ’16, 31(5).

**Acknowledgements** Ahmed and Bagherzadeh were supported in part by Oakland University Department of Computer Science and Research Office.