Synthesizing Program Execution Time Discrepancies in Julia Used for Scientific Software

Effat Farhana
Dept. of Computer Science
North Carolina State University
efarhan@ncsu.edu

Nasif Imtiaz
Dept. of Computer Science
North Carolina State University
simtiaz@ncsu.edu

Akond Rahman
Dept. of Computer Science
Tennessee Tech. University
akond.rahman.buet@gmail.com

***Abstract*—Scientific software is defined as software that is used to analyze data to investigate unanswered research questions in the scientific community. Developers use programming languages such as Julia to build scientific software. When programming with Julia, developers experience program execution time discrepancy i.e. not obtaining desired program execution time, which hinders them to efficiently complete their tasks. *The goal of this paper is to help developers in achieving desired program execution time for Julia by identifying the causes of why program execution time discrepancies happen with an empirical study of Stack Overflow posts*. We conduct an empirical study with 263 Julia-related posts collected from Stack Overflow, and apply qualitative analysis on the collected 263 posts. We identify 9 categories of program execution time discrepancies for Julia, which include discrepancies related to data structures usage such as, arrays and dictionaries. We also identify 10 causes that explain why the program execution time discrepancies happen. For example, we identify program execution time discrepancy to happen when developers unnecessarily allocate memory by using array comprehension.**

***Index Terms*—Julia, programming language, stack overflow**

I. Introduction

Scientific software is defined as software that is used to explore and analyze data to investigate unanswered research questions in the scientific community [1]. The domain of scientific software includes software needed to construct a research pipeline such as software for simulation and data analysis, large-scale dataset management, communication infrastructure, and mathematical libraries [2] [3]. Programming languages such as Julia [4] are used to develop scientific software efficiently and achieve desired program execution time. Julia was used in Celeste [1], a software used in astronomy research. Celeste was used to load 178 terabytes of astronomical image data to produce a catalog of 188 million astronomical objects in 14.6 minutes, yielding a program execution time improvement by a factor of 1,000, compared to prior implementation [2]. The Celeste-related example provides an anecdotal evidence on how Julia could be beneficial to achieve desired program execution time, and complete computations tasks efficiently.

However, not all developers obtain the perceived benefits of Julia with respect to program execution time. On online forums developers have reported program execution time discrepancies i.e., not achieving desired program execution time for a program written in Julia. For example, one developer reported to be "giving up on Julia" because the developer observed "a trivial hello world program" to respectively, run 27 times and 187 times slower in Julia compared to that of Python and C [3]. The above-mentioned anecdotal evidence suggests discrepancy in program execution time for developers who use Julia, which may hinder developers in completing their tasks efficiently. The Carnegie Mellon Software Engineering Institute identified discrepancies in program execution time as a risk for scientific software development, which should be mitigated by the stakeholders [5].

Systematically investigating the discrepancies in program execution time for Julia could derive clues to help developers in achieving expected program execution time for software programs written in Julia. Let us consider Figure 1 in this regard. In Figure 1, we present an excerpt from a Stack Overflow (SO) post [4], where a SO user reports a Julia program to run 44 times slower than Fortran (Figure 1a). According to Figure 1b, two causes are identified: (i) global variables; and (ii) array slicing [4]. Evidence from Figure 1 suggests by analyzing SO posts we can identify what Julia-related program execution time discrepancies developers face, and the corresponding causes.

The goal of this paper is to help developers in achieving desired program execution time for Julia by identifying the causes of why program execution time discrepancies happen with an empirical study of Stack Overflow posts.

We answer the following research questions (RQs):

- **RQ1**: *What categories of program execution time discrepancies for Julia are reported by developers in Stack Overflow posts? How frequently do identified categories of program execution time discrepancies appear?*
- **RQ2:** *What causes explain program execution time discrepancy for Julia?*

We collect 263 Julia-related posts from Stack Overflow (SO) to conduct our empirical study. We apply a qualitative analysis technique called descriptive coding [6] to determine (i) the categories of program execution time discrepancies, and (ii)

[1]https://www.hpcwire.com/off-the-wire/julia-joins-petaflop-club/
[2]https://juliacomputing.com/case-studies/celeste.html
[3]http://www.zverovich.net/2016/05/13/giving-up-on-julia.html
[4]https://stackoverflow.com/questions/20613817/

2576-3148/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSME.2019.00083

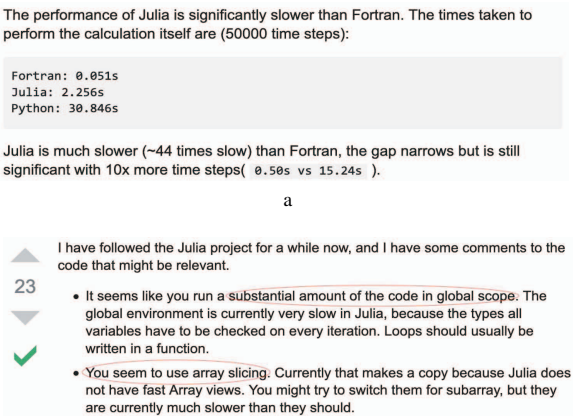

Fig. 1: Example SO Post Related to Program Execution Time Discrepancy. Figures 1a and 1b respectively, identifies a discrepancy in program execution time, and the corresponding cause.

the causes that explain program execution time discrepancies that occur in Julia programs.

We list our contributions as following:

- A list of categories for Julia-related program execution time discrepancies;
- A list of causes that explain program execution time discrepancy for Julia programs; and
- An analysis of how frequently the identified categories and causes appear in SO.

We organize the rest of the paper as following: we describe the methodology of the paper in Section II. We describe our findings in Section III. We discuss our findings in Section IV and related prior research in Section V. We conclude the paper in Section VI.

II. Methodology

In SO, users can post questions that describe a specific problem that they want to seek advice on [7]. Each question has a title that provides a concise summary of what the question is about [7]. The details of the question is presented in the body, where users can describe the problem in detail with additional references [7]. Each question has one or many tags, which is used to identify the applicable language or technology for the question.

We use the SOTorrent dataset [8] to collect Julia-related posts. First, we identify SO questions with the tag *'julia-lang'* to extract Julia related SO questions. According to prior work [9] [10], SO datasets suffer from quality issues. Similar to prior research [10], we apply a filtering criteria to improve quality of the downloaded data, which is summarized in Table I. We investigated our analysis of two RQs based on these 263 posts.

RQ1: What categories of program execution time discrepancies for Julia are reported by developers in Stack Overflow posts? How frequently do identified categories of program execution time discrepancies appear?

TABLE I: Selection of Julia-related Stack Overflow Posts for Analysis

Initial post count	41,782,536
Criteria-1 (Ques. with at least one answer)	14,207,037
Criteria-2 (Ques. with score > 0)	6,902,332
Criteria-3 (Ques. tagged as 'julia-lang')	3,150
Criteria-4 (Ques. with 'performance' and 'optimization')	533
Criteria-5 (Ques. filtered manual inspection)	263
Final question count	263

To answer the first part of RQ1, we analyzed the question that describe a specific query related to program execution time for Julia implementation. From the textual content of the questions, we apply qualitative analysis technique called descriptive coding [6]. First, we read the question description and title to obtain raw text, which are merged into codes. Next, we merged the codes based on similarities to derive categories.

In our analysis, one SO post can belong to multiple categories. The first author derived the categories, which is susceptible to bias. We mitigate the first author's bias by sending a random sample of 25 posts to the third author for reviewing. We calculate F1-measure for agreement rating between the first and third author similar to Chen et al. [11]. In our analysis one SO post can belong to multiple categories, and common techniques such as Cohen's Kappa are not well-suited. F1-measure reports harmonic mean to represent how close two label sets are assigned to one SO post by two raters [12]. A zero score indicates complete different and an one represents complete identical rating. The recorded F1-score to determine categories was 0.6.

To answer the second part of RQ1, we report the normalized frequency of questions (Q(x)), answer to question ratio (AQ(x)), and normalized view count (VQ(x)). We use these three metrics as these three metrics have been used to determine frequency analysis of SO posts [13] [14].

RQ2: What causes explain program execution time discrepancy for Julia?

To identify the causes for program execution time discrepancy, we applied descriptive coding similar to RQ1 on comments and answers providing solution. Similar to our RQ1 analysis, a post may have multiple causes for discrepancies related to program execution time. The first author listed 10 causes by applying descriptive coding on 263 posts. To mitigate bias introduced by the first author, the third author applied qualitative analysis on a set of 25 posts, and identify mapping between the 25 posts and identified reasons. The F1-score between the first and third author was 0.6.

III. Empirical Findings

In this section we present our findings.

A. Answer to RQ1

What categories of program execution time discrepancies for Julia are reported by developers in Stack Overflow

posts?: We identify nine categories of program execution time discrepancies for Julia. The number of SO posts for each category is shown in parenthesis in Table II. Each post can belong to multiple categories. We describe each category with explanations and example SO posts. Throughout the rest of this paper, references to example SO posts are presented in the format of '($POSTID$)'.

1) Data Structure: This discrepancy is related to program execution time discrepancies that happens for Julia data structures usage. The design, analysis, and implementation of data structures determine program execution time [15]. With respect to frequency, this category is the largest. Users asked questions about array, vector, dictionary, and dataframe manipulation. In an example post a developer asks faster summation over an array (36801197).

2) Mathematics/Statistics/Machine Learning: This discrepancy is related to mismatches in program execution time for mathematical calculations, which is prevalent amongst numeric operations and machine learning tasks [4]. For example, a developer sought solution for the fastest way to compute the sum of outer products of an n-dimensional column vector (38773076).

3) Language Transfer: This discrepancy happens when developers transfer to Julia from another programming language. Transfer learning of programming language poses confusion in adopting a new language [16] and developers do not observe the expected speedup in Julia. An example SO post expresses user's dissatisfaction with Julia's performance comparing with Matlab to compute eigenvectors of large matrices (40034479).

4) Function/Package/Compilation: This discrepancy is related to program execution time discrepancies that happens in compile time for Julia. This category also includes compilation time of Julia functions and packages. In an example post we see question about `eval` function usage from a dictionary (41639237).

5) Parallel Programming: This discrepancy is related to program execution time discrepancies for software using Julia's parallel programming features. Julia's built-in primitives for parallel computing [17] have been deployed for speeding up computational tasks [18]. In an example post we notice user wants to speed up gradient descent calculation by parallelization (31656858).

6) Data Type Concern: This discrepancy is related to program execution time discrepancies that occurs for Julia's data types. Data type declaration is related to memory allocation and program's execution time [19]. This category documents posts with type concerns. A developer in an example post asks about the efficient implementation of Julia template (45972534).

7) Memory Allocation: This discrepancy is related to program execution time discrepancies that occurs due to inefficient allocation of memory for Julia programs. Programs' execution time are determined by memory allocation by using data structure and types [19]. In an example SO post, we observe user's concern about huge memory allocation of the program (38399478).

TABLE II: Categories of Program Execution Time Discrepancies with Frequency Analysis. Q = Normalized Question Frequency, AQ = Answer to Question Ratio, and VQ = Normalized View Count.

Category	Q	AQ	VQ
Data Structure Concern (103)	0.36	1.68	556.38
Mathematics/Statistics/Machine Learning (57)	0.20	1.48	384.43
Language Transfer (42)	0.15	1.86	1678.12
Function/Package/Compilation (36)	0.13	1.39	581.50
Parallel Programming (34)	0.12	1.41	580.09
Data Type Concern (21)	0.07	1.52	239.00
Memory Allocation (16)	0.06	1.69	237.56
File Operation (12)	0.04	1.25	454.67
Scope Concern (7)	0.02	1.57	288.71

8) File Operation: This discrepancy is related to program execution time discrepancies that occurs during reading and writing datasets. As data storage and access are important for scientific tasks, efficient file operation is required to avoid program execution time discrepancies [20]. An example SO post question discusses about efficient implementation of file input to read a matrix (26810171).

9) Scope Concern: This discrepancy is related to the scope of variables declared in Julia. Global variables slow down Julia's execution time [21]. Branches, loop, and control flow shape a program's execution time behavior [19]. An example post of this category is (29877563).

How frequently do identified categories of program execution time discrepancies appear?: Table II lists the Q (x), AQ (x), and VQ (x) for all developed categories. The Q (x) count is related with the number of posts belonging to each category. From Table II, we observe the highest view count for the category 'Language Transfer', which suggests the prevalence of this category of discrepancy amongst both registered and non-registered SO users.

B. Answer to RQ2

In this section, we answer ***: What causes explain program execution time discrepancy for Julia?*** We describe the causes associated with program execution time discrepancy for Julia. Each cause is presented below with examples from SO presented in the format of '($POSTID$)'.

1) Inefficient Program Construct: This category is indicative of syntax-related causes for program execution time discrepancy. The post answer provides the syntax of right or efficient implementation package and function. An example post answer suggests the use of 'readdlm' [4] to read a file efficiently, instead of using a loop to read the file character by character (26810171).

2) Memory Management Knowledge Gap: This cause includes degradation of program execution time due to memory allocation. The Julia documentation for performance recommends users to avoid unwanted memory allocation [21]. This

can be done by using in place version of functions (i.e., in-place version '$sort$!' instead of '$sort$'), declaring variable outside loop by manual inspection, avoiding array comprehension, array slicing, and unnecessary temporary variables. An accepted answer suggests to rewrite code by allocating array outside the loop for efficient dot product calculation (38687435).

3) Julia Concepts Knowledge Gap: This cause includes understanding Julia-specific features such as, longer execution time for first run in Julia due to just-in-time compilation (47501844), accessing arrays in column major order for faster manipulation (29742768). The Julia documentation page for performance tip lists these two features for correct time comparison and performance enhancement [21].

4) Lack of Computer Science Fundamentals: Developers experience discrepancies in program execution because they don't have necessary knowledge in algorithm-related concepts such as time complexity. For example, a developer was unaware on the effects of using nested loops, which resulted in $O(n^2)$ time complexity. Replacing the $O(n^2)$ implementation with $O(n)$ resulted in performance boost (31321810).

5) Global Scope: Developers experience program execution time discrepancies due to lacking knowledge of Julia's scope. Code constructs such as variables used in a global scope can change at any point during program execution, forcing the compiler to decide the types of the variables in runtime, leading to slower program execution time in Julia [21]. An example SO post answer suggests performance boosting by avoiding declaring global variables and wrap up code fragments in functions, avoiding extra memory allocation, and writing explicit for loops instead of vectorization (20613817).

6) Type Instability: One identified cause for program execution time discrepancy is type-instable code. Type-instablity arises if the compiler needs to decide data type of variables during runtime. The Julia documentation for performance tips [21] lists to avoid type-instable practices such as abstract data type, anonymous function, and mismatching of return type in a function. In a SO post, one suggested solution is to declare variables with appropriate types such as Double or Integer (42043590).

7) Parallel Programming Knowledge Gap: This cause lists poor program execution time due to lack of parallel computing concepts. Examples include: not using parallel programming specific data structure such as, `SharedArray` or `DistributedArray` (31656858), starting more processes than cores (50975707), and not using '$@everywhere$' to pass data among worker threads (26168943).

8) Vectorization: The vectorization form in Julia requires memory allocation. Vectorization involves of operations that operate on whole arrays. For example, $x+ = 0.5 * (y + z)$ is an example of vectorization, where x, y, and z are arrays. We notice suggestion to explicitly use a for loop, instead of using vectorization (20613817).

9) Macro Concept Gap: We observe developers not to be aware of Julia's macros to achieve desired program execution time. Macros offer program execution time boosting by operating on expressions that will be compiled once. Macros are part of Julia's metaprogramming feature and listed in Julia docs for speedup [22]. Examples suggesting macros are: using `@inbound` with for loops to avoid bound checking (25009072), replacing `eval` by `@eval` (20174352).

10) Data Structure Knowledge Gap: This cause includes not using appropriate data structure implementation to achieve better program execution time. An example post labeled with the category "Data Structure Concern" and "Mathematics/Statistics/Machine Learning" has asked efficient implementation of Markov Chain (50274934). The accepted answer suggests using view and work on a transpose of a matrix.

The number of SO posts that belong to each cause is presented in Figure 2. A SO post where program execution time discrepancy is reported, can be explained using one or more causes.

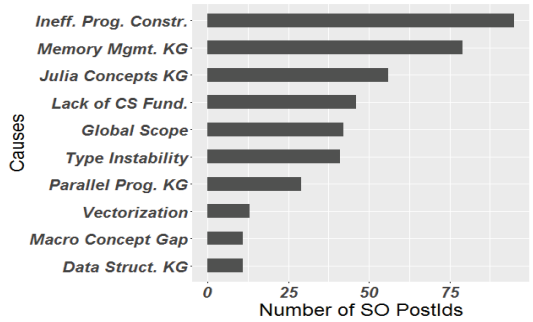

Fig. 2: Program execution time discrepancy causes with frequency. KG stands for knowledge gap.

IV. Discussion

Based on our findings from Section III-A we recommend developers to use Julia-specific programming constructs to diagnose discrepancies related to program execution time they experience. Examples of such constructs are: using '$@code\_warntype$' to diagnose type instability [23], '$@time$' and '$@allocated$' to measure memory allocation with time spent [21], and '$@profile$' to investigate time spent by each line [23].

The identified causes from Section III-B are mentioned by the Julia community in forms of community guidelines [21] and documentation [23]. Answers to RQ2 suggest that developers may not be reading the community guidelines, and relying on the SO community, as suggested by Parnin et al. [24]. We recommend developers to consult the Julia documentation and Julia performance guidelines when writing Julia programs. As SO is popular amongst developers for seeking guidance, the Julia community can identify strategies to be involved more with SO community, and resolve developers issues.

Limitations of Our Study: The identified categories and causes are susceptible to the bias of the first author. We

mitigate this limitation by assigning a rater to categorize a subset of the SO posts. We only use SO posts, which could suffer from external validity. In future, we plan to mitigate this limitation by including questions and answers from the Julia Discourse Forum [25].

V. Related Work

Our paper is related to prior research studies in the domain of scientific software. Milewicz et al. [3] studied how collaboration happens amongst developers while developing open source scientific software. Howison and Herbshleb [26] identified incentives for creating and maintaining scientific software. Bangherth and Heister [27] identified what practices are helpful to develop open source scientific software libraries. The above-mentioned prior research did not address what categories of performance discrepancies developers face when developing scientific software.

Prior research has extensively used SO datasets to gather insights for different domains. Examples include studying collaborative learning features [28], classifying expert users [29], analyzing Java security [9], analyzing Python-related security [10], and analyzing static analysis alert resolution [13]. The above-mentioned discussion suggests that studying SO posts could be useful for domain-specific research such as Julia.

VI. Conclusion

In the domain of scientific software, achieving desired program execution time is vital for developers, as the developed software is involved in computationally intensive scientific applications. Discrepancies related to program execution time for scientific software could hinder developers in completing their tasks. We apply qualitative analysis with 263 Julia-related SO posts, and identify 9 categories of program execution time discrepancies. We also identify 10 causes, which explain program execution time discrepancies. Based on our analysis, we provide two recommendations: (i) use documentation to follow Julia coding best practices, and (ii) use program profiling to diagnose program execution time discrepancies.

We hope that future research will investigate other sources of data, such as developer surveys, and the Julia Discourse Forum [25]. Researchers can explore other languages used in scientific software to investigate if other categories of program execution time discrepancies are identified. We hope our paper can facilitate further research in the domain of Julia and scientific software.

References

[1] E. S. Mesh and J. S. Hawker, "Scientific software process improvement decisions: A proposed research strategy," in *2013 5th International Workshop on Software Engineering for Computational Science and Engineering (SE-CSE)*, May 2013, pp. 32–39.

[2] G. T. Jeffrey. Carver, Neil Hong, *Software Engineering for Science*, 1st ed. NY, NY, USA: CRC Press, 2016.

[3] R. Milewicz, G. Pinto, and P. Rodeghero, "Characterizing the roles of contributors in open-source scientific software projects," in *Proceedings of the 16th International Conference on Mining Software Repositories*, ser. MSR '19, 2019, pp. 421–432.

[4] "The Julia language," https://docs.julialang.org/en/v1/.

[5] R. Kendall, D. Post, J. Carver, D. Henderson, and D. Fisher, "A proposed taxonomy for software development risks for high-performance computing (hpc) scientific/engineering applications," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2006-TN-039, 2007.

[6] J. Saldaña, *The coding manual for qualitative researchers*. Sage, 2015.

[7] S. Exchange, "Stack Exchange," https://data.stackexchange.com/, 2019, [Online; accessed 08-06-2019].

[8] S. Baltes, L. Dumani, C. Treude, and S. Diehl, "Sotorrent: Reconstructing and analyzing the evolution of stack overflow posts," in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18, 2018, pp. 319–330.

[9] N. Meng, S. Nagy, D. Yao, W. Zhuang, and G. Arango-Argoty, "Secure coding practices in java: Challenges and vulnerabilities," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 372–383.

[10] A. Rahman, E. Farhana, and N. Imtiaz, "Snakes in paradise?: Insecure python-related coding practices in stack overflow," in *Proceedings of the 16th International Conference on Mining Software Repositories*, ser. MSR '19, 2019.

[11] X. Chen, M. Vorvoreanu, and K. Madhavan, "Mining social media data for understanding students learning experiences," *IEEE Transactions on Learning Technologies*, vol. 7, no. 3, pp. 246–259, 2014.

[12] J. L. Devore, *Probability and Statistics for Engineering and the Sciences*. Cengage Learning, 2015.

[13] N. Imtiaz, A. Rahman, E. Farhana, and L. Williams, "Challenges with responding to static analysis tool alerts," in *Proceedings of the 16th International Conference on Mining Software Repositories*, ser. MSR '19, 2019.

[14] A. Rahman, A. Partho, P. Morrison, and L. Williams, "What questions do programmers ask about configuration as code?" in *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering*. ACM, 2018, pp. 16–22.

[15] M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, *Data structures and algorithms in Java*. John Wiley & Sons, 2014.

[16] N. Shrestha, T. Barik, and C. Parnin, "It's like Python but: Towards supporting transfer of programming language knowledge," in *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2018, pp. 177–185.

[17] "Julia and parallel computing," https://juliacomputing.com/domains/parallel-computing.html.

[18] "Julia featured by Intel," https://juliacomputing.com/blog/2017/07/25/julia-in-intel-parallel-universe.html.

[19] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1986.

[20] J. Li, W.-k. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, "Parallel netcdf: A high-performance scientific i/o interface," in *SC'03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*. IEEE, 2003, pp. 39–39.

[21] "Performance tips: Juliadoc," https://docs.julialang.org/en/v1/manual/performance-tips/index.html.

[22] "Metaprogramming in Julia: Macros," https://docs.julialang.org/en/latest/manual/metaprogramming/.

[23] "Julia code profiling," https://docs.julialang.org/en/v1/manual/profile/index.html.

[24] C. Parnin and C. Treude, "Measuring api documentation on the web," in *Proceedings of the 2nd international workshop on Web 2.0 for software engineering*. ACM, 2011, pp. 25–30.

[25] "Julia discourse forum," https://discourse.julialang.org/.

[26] J. Howison and J. D. Herbsleb, "Scientific software production: Incentives and collaboration," in *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*, ser. CSCW '11. New York, NY, USA: ACM, 2011, pp. 513–522. [Online]. Available: http://doi.acm.org/10.1145/1958824.1958904

[27] W. Bangerth and T. Heister, "What makes computational open source software libraries successful?" *Computational Science & Discovery*, vol. 6, no. 1, p. 015010, nov 2013.

[28] J. Loeckx, "Mining for evidence of collaborative learning in question & answering systems." in *EDM (Workshops). 2014*.

[29] D. Posnett, E. Warburg, P. Devanbu, and V. Filkov, "Mining stack exchange: Expertise is evident from initial contributions," in *Social Informatics (SocialInformatics), 2012 International Conference on*. IEEE, 2012, pp. 199–204.