

Parallel Test Case Prioritization for Distributed System Using Search Algorithms

Team 6

Seyoung Song, Subeom Park, Yoonho Nam, Azret Kenzhaliev

Abstract

Abstract (by Seyoung)

Contents

1	Introduction	1
2	Problem Description (by Subeom)	2
3	Algorithms	2
3.1	Greedy Algorithms (by Azret)	2
3.2	Simulated Annealing (by Subeom)	2
3.3	Genetic Algorithms (by Yoonho)	2
4	Experimental Design (by Seyoung)	2
4.1	Research Questions	2
4.2	Subjects	3
4.3	Settings	4
5	Evaluation Results	5
5.1	Greedy Algorithms (by Azret)	5
5.2	Simulated Annealing (by Subeom)	5
5.3	Genetic Algorithms (by Yoonho)	5
5.4	Comparison (by Seyoung)	5
6	Conclusion	5
	References	6

1 Introduction

- Regression Test Case Prioritization
- Parallel Test Prioritization
- Parallel Test Prioritization, but different CPU

2 Problem Description (by Subeom)

TODO: describe the problem (Parallel Test Prioritization)

- Problem Description
- Problem Definition
- Effectiveness Measure

3 Algorithms

TODO: describe your approach

3.1 Greedy Algorithms (by Azret)

3.2 Simulated Annealing (by Subeom)

3.3 Genetic Algorithms (by Yoonho)

4 Experimental Design (by Seyoung)

4.1 Research Questions

- RQ1: Which algorithm is most effective in solving the parallel test prioritization problem?
- RQ2: How do the number of computing resources and the relative performance between them influence the performance of the parallel test prioritization techniques?

4.2 Subjects

Table 1: Open-source subjects from GitHub

ID	Subjects	SLOC	#Test	Time (s)
1	commons-cli	9053	192	3.064
2	dictomaton	4318	53	14.067
3	disklrucache	1921	61	2.364
4	efflux	5633	40	0.581
5	exp4j	5699	311	11.350
6	gdx-artemis	3607	35	0.483
7	geojson-jackson	1569	60	1.284
8	gson-fire	3566	91	3.249
9	jactor	6984	60	11.628
10	jadventure	5276	74	2.311
11	jarchivelib	2256	33	0.217
12	java-faker	8541	571	34.154
13	java-uuid-generator	4321	46	0.937
14	javapoet	9874	346	15.323
15	jsonassert	3476	150	1.641
16	jumblr	2970	103	0.905
17	lastcalc	7271	34	13.672
18	low-gc-membuffers	13099	51	1.784
19	metrics	6493	76	43.964
20	mp3agic	10037	495	4.815
21	nv-websocket-client	8617	73	1.014
22	protoparser	5545	171	4.752
23	restfixture	8243	290	6.716
24	skype-java-api	9749	24	15.720
25	stateless4j	2728	88	2.146
26	stream-lib	8756	142	443.206
27	xembly	3030	63	6.834

4.3 Settings

1. Sequential Test Prioritization
 - $c = \{1\}$
2. Parallel Test Prioritization
 - $c = \{2, 4, 8, 16\}$
3. Asymmetric Test Prioritization
 - $1 : 2$
 - $1 : 3$
 - $1 : 4$
 - $1 : 1 : 1 : 1 : 4 : 4 : 4 : 4$

Table 2: An example of computing scenarios

Computing Scenario	Relative Performances
Sequential Test Prioritization	[1]
Parallel Test Prioritization ($c = 2$)	[1, 1]
Parallel Test Prioritization ($c = 4$)	[1, 1, 1, 1]
Parallel Test Prioritization ($c = 8$)	[1, 1, 1, 1, 1, 1, 1, 1]
Parallel Test Prioritization ($c = 16$)	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Asymmetric Test Prioritization ($1 : 2$)	[1, 2]
Asymmetric Test Prioritization ($1 : 3$)	[1, 3]
Asymmetric Test Prioritization ($1 : 4$)	[1, 4]
Asymmetric Test Prioritization ($1 : 1 : 1 : 1 : 4 : 4 : 4 : 4$)	[1, 1, 1, 1, 4, 4, 4, 4]

Table 3: Settings for additional experiments

Performance of Computing Resources p
Sequential ($c = 1$)
Parallel ($c = 2$)
Parallel ($c = 4$)
Asymmetric ($1 : 3$)
Asymmetric ($1 : 1 : 2$)
Asymmetric ($2 : 2$)
Parallel ($c = 8$)
Asymmetric ($1 : 7$)
Asymmetric ($2 : 6$)
Asymmetric ($3 : 5$)
Asymmetric ($4 : 4$)
Asymmetric ($2 : 2 : 2 : 2$)
Asymmetric ($1 : 1 : 3 : 3$)
Parallel ($c = 12$)
Parallel ($c = 25$)
Parallel ($c = 50$)
Parallel ($c = 100$)
Parallel ($c = 200$)

Table 4: Settings for additional experiments

Time Constraint TC
1.25
1.5
1.75
2.0

5 Evaluation Results

TODO: describe the evaluation results

5.1 Greedy Algorithms (by Azret)

5.2 Simulated Annealing (by Subeom)

5.3 Genetic Algorithms (by Yoonho)

5.4 Comparison (by Seyoung)

Setting	GA	SA	AGA
Sequential	0.932	0.868	0.876
Parallel ($c = 2$)	0.964	0.936	0.937
Parallel ($c = 4$)	0.980	0.966	0.966
Parallel ($c = 8$)	0.987	0.981	0.981
Parallel ($c = 16$)	0.990	0.988	0.988
Asymmetric (1 : 2)	0.976	0.955	0.958
Asymmetric (1 : 3)	0.982	0.963	0.968
Asymmetric (1 : 4)	0.985	0.968	0.975
Asymmetric (1 : 1 : 1 : 1 : 4 : 4 : 4 : 4)	0.995	0.992	0.993

6 Conclusion

- Discussion
 - Comparison With Sequential Test Prioritization
 - Practical Concerns
 - Generalizability
- Comments from Professor
 - How long should the entire test take for there to be real gains in prioritization?
 - The time gain from prioritization becomes smaller as the number of compute resources increases, so it may not be meaningful if you already have a lot of compute resources.

References: [1]–[4]

References

- [1] Z. Li, M. Harman, and R. M. Hierons, “Search algorithms for regression test case prioritization,” *IEEE Trans. Software Eng.*, vol. 33, no. 4, pp. 225–237, Apr. 2007, doi: 10.1109/TSE.2007.38.
- [2] J. Chen *et al.*, “Optimizing test prioritization via test distribution analysis,” in *Proceedings of the 2018 26th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, Oct. 2018, pp. 656–667. doi: 10.1145/3236024.3236053.
- [3] Q. Luo, K. Moran, L. Zhang, and D. Poshyvanyk, “How do static and dynamic test case prioritization techniques perform on modern software systems? An extensive study on GitHub projects,” *IEEE Trans. Software Eng.*, vol. 45, no. 11, pp. 1054–1080, Nov. 2019, doi: 10.1109/TSE.2018.2822270.
- [4] J. Zhou, J. Chen, and D. Hao, “Parallel test prioritization,” *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 1, Sep. 2021, doi: 10.1145/3471906.