13th International Conference on Applied Informatics
Eszterházy Károly Catholic University
Eger, Hungary, February 19–21, 2026

Parallel Approaches of Annotation-Based Static Verification of Algorithmic Complexity in Java

Aleksandr Samedov[a], **Tamas Kozsik**[b]

[a]ELTE Eötvös Loránd University, Faculty of Informatics
emsmx4@inf.elte.hu ORCID:0009-0008-1853-0292

[b]ELTE Eötvös Loránd University, Faculty of Informatics
kto@inf.elte.hu* ORCID:0000-0003-4484-9172

Abstract

Static analysis tools must handle increasingly large software projects, yet their performance is often limited by sequential execution. This paper explores three parallelism levels for annotation checking: (1) file-level, (2) annotation-level, and (3) AST-level. Comparing these on benchmarks like Elasticsearch and Spring Boot, we evaluate scalability and synchronization costs. Results show file-level parallelism is effective for large projects, while batched annotation-level parallelism offers the best balance for dense annotations. We conclude that a hierarchical strategy is the most practical path to efficient static analysis.

1. Introduction

Verifying algorithmic complexity via annotations (e.g., `@Prove("O(n)")`) is critical for preventing performance regressions. However, sequential static analysis creates bottlenecks in CI/CD environments as codebases grow.

This work extends our previous sequential verification framework [6] by introducing and evaluating parallel execution models. We investigate three distinct levels of parallelism:

*This work was supported by project no. TKP2021-NVA-29 under the Ministry of Innovation and Technology of Hungary from the National Research, Development, and Innovation Fund and financed under the TKP2021-NVA funding scheme.

1. **File-Level:** Coarse-grained processing of independent source files.
2. **Annotation-Level:** Concurrent analysis of annotated elements within a file.
3. **AST-Level:** Fine-grained parallel traversal of AST subtrees.

We evaluate these on Elasticsearch [1], Spring Boot [8], and libGDX [2] to determine optimal granularity.

2. Methodology

File-Level Parallelism treats each `.java` file as an independent unit managed by a fixed-size `ExecutorService`. This offers low synchronization costs but suffers from imbalance if "God classes" dominate execution.

Annotation-Level Parallelism parses files sequentially but processes annotated elements concurrently. We implemented a "Batched" variant (20–40 methods/task) to mitigate `ForkJoinPool` overhead, beneficial for dense annotation usage [5].

AST-Level Parallelism targets intensive analysis of single methods by traversing AST branches in parallel. While theoretically attractive, scheduling overhead often outweighs gains for standard verification.

3. Evaluation

Experiments used a Dell Latitude 7430 (i5–1245U, 10 cores, 32 GB RAM) with OpenJDK 17. We recorded the minimum wall-clock time over 10 runs to filter noise [4].

Elasticsearch: File-level parallelism provided 1.79x speedup (4 threads). However, the "Batch 40" strategy yielded the highest speedup of **1.95x** (Table 1).

Table 1. Elasticsearch Performance (ms).

Strategy	Seq. Time	T_4 Time	T_4 Speedup	T_{12} Speedup
File-level	19624	12023	1.79x	1.72x
Annotation-level	19624	12510	1.74x	1.57x
Batch 40	19624	11153	**1.95x**	1.88x

Spring Boot: The "Batch 20" strategy achieved a peak speedup of **1.90x**, outperforming pure File-level (1.82x). Speedup plateaued beyond 6 threads across all benchmarks, consistent with Amdahl's Law due to I/O and class loading limits [3].

4. Conclusion

A "one-size-fits-all" strategy is suboptimal. While File-level parallelism is a robust baseline, it fails on complex individual files. Conversely, fine-grained AST

parallelism incurs prohibitive overhead [7]. We propose a "Dynamic Hierarchical Strategy": default to File-level, but dynamically switch to Batched Annotation-level tasks if a file exceeds a processing threshold.

References

[1] Elastic: *Elasticsearch: Open Source, Distributed, RESTful Search Engine*, [Source code]. GitHub, 2023, url: https://github.com/elastic/elasticsearch.

[2] B. Games: *libGDX: Desktop/Android/HTML5/iOS Java Game Development Framework*, [Source code]. GitHub, 2023, url: https://github.com/libgdx/libgdx.

[3] B. Goetz, T. Peierls, J. Bloch, J. Bowbeer, D. Holmes, D. Lea: *Java Concurrency in Practice*, Pearson Education, 2006.

[4] A. H. Karp, H. P. Flatt: *Measuring parallel processor performance*, Communications of the ACM 33.5 (1990), pp. 539–543.

[5] D. Lea: *A Java fork/join framework*, Proceedings of the JAVA Grande Conference (2000), pp. 36–43.

[6] A. Samedov, Z. Porkolab: *Annotation-Based Static Verification of Algorithmic Complexity in Java*, in: Proc. 12th Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications (SQAMIA), vol. 4077, CEUR Workshop Proceedings, 2025, Paper 16, url: https://ceur-ws.org/Vol-4077/paper16.pdf.

[7] V. Subramaniam: *Programming Concurrency on the JVM: Mastering Synchronization, STM, and Actors*, Pragmatic Bookshelf, 2011.

[8] I. VMware: *Spring Boot*, [Source code]. GitHub, 2023, url: https://github.com/spring-projects/spring-boot.