Thesis title**: Clustering Analysis of Lock Contention Fault Types Using Run-Time Performance Metrics for Java Intrinsic Locks**
Candidate**: Nahid Hasan Khan**
Supervisors**: Dr. Akramul Azim (supervisor), and Dr. Ramiro Liscano (co-supervisor).**
A review done by the Supervisory Committee member: **Dr. Sanaa Alwidian, FEAS**

Contribution of the thesis to advancing knowledge

The thesis brings the following main original contributions:

1. Classifying contention fault types of java-based concurrent application through clustering techniques utilizing the run-time metrics that come from performance analyzer tools.
2. Generation of a dataset containing contention statistics and formalization of the experiments so that by leveraging this formalization one can enrich dataset with new sets of contention faults.

Overall:
1. Very well-written, organized, and easy to follow and understand thesis.
2. The candidate showed a great in-depth and in-breadth understanding of the subject matter.
3. The Literature Review chapter is very good,), which helps to put into context the original contributions of the thesis.
4. My comments are minor and of clarification nature.

Revisions necessary for thesis to be accepted

No major revisions are necessary. Some questions will be asked during the defense. These questions are intended to clarify some minor details of the thesis.

In the meanwhile, I have the following comments that I would like the candidate to address (for an easier tracking, please consider my comments highlighted in Yellow at the thesis PDF file):

My comments:

1. Page 2 (Section 1.1-Introduction): The focus of this thesis is on **Lock contention**, which typically happens when *thread A* has a lock and *thread B* wants to acquire that same lock, thread B will have to wait until thread A releases the lock. While I agree that the most obvious example of contention is on a lock), but have you thought of scenarios when "locks" are not the only resources on which contention can be experienced?
   For example, thread *A* may experience slowdowns even if it never has to wait for the other threads (say thread B) to release the lock! This is because a lock protects some kind of data, and the data itself will often be contended as well.

2. Based on comment #1, I would like to see a brief discussion about the contention problem in general, whether it is caused by “locks” or when it just happened simply when two threads try to access either the same resource or related resources in such a way that at least one of the contending threads runs more slowly than it would if the other thread(s) were not running.

3. Page 3: It was mentioned that “…Therefore, we believe that it is possible to classify contention fault types into these two causes using a clustering approach that will identify the essential features from the JLM and perf run-time metrics”. Again, what if there are other “non-lock-driven” contentions, how would the classifier behave in this case?

4. Page 4: It was mentioned that “In our approach, we create a dataset considering different features that are responsible for creating these faults”. How scalable is this dataset? what if new features emerged? is there a kind of “combinatorial” features that could cause contention faults? by combinatorial features I mean of you combine two or more features (where each feature per se is not a problem), their combination could cause a problem.

5. Page 5: It was mentioned that “Multiple threads have the permission to read simultaneously as long as any thread does not attempt to write at that moment, or there is no incoming write request”. What if after granting a read permission, there becomes lots of incoming write requests? Will this a contention fault again?

6. Page 8: According to the suggested solution #4, how would implementing the read-write lock offer a solution for reducing the lock competition and sending access requests to the locks? please explain.

7. Page 12: I assume that Peter Hover et al., [18] have identified the same contention causes that you've identified. Other than the metrics, what is the difference between their approaches to solve each type of contention, and your proposed approach? Please explain further.

8. Page 29, It is mentioned that “we emulated with 4 threads and applied 1 millisecond inside the critical section”. Why 1 millisecond in particular? you always need to provide a justification/explanation regarding the settings of your experimental parameters.

9. Page 30: It was mentioned that “Although we are interested in two classes referring to the two potential faults, our generated data should contain more categories than the two”. What are these more categories? Please give examples.

10. Page 30 and Figure 3.1: It was mentioned that “These steps are shown in a high-level workflow (see Figure 3.1)…”. In the high-level workflow figure (Figure 3.1), I suggest that you illustrate explicitly (using big, dashed boxes) the three main steps (1. acquiring run-time metrics, 2. Filtering of metrics, 3. data processing and classification) on top of the detailed sub-steps that you already had in the figure.

11. Page 31: It was mentioned that “It is ideal that the data we fed into an ML model should be streamed from a single source rather than multiple files”. Again, you need to explain the rational behind your choices. Why a single source rather than multiple files?

12. Page 31: It was mentioned that “Before analyzing the data using pure KMeans, we preprocess our data, scale, and reduce the features leveraging Principal Component Analysis (PCA)”. What are the exact criteria of data processing you used in order to get more improved data? For example, what were the feature reduction criteria?

13. Page 31: “First, we collect performance metrics data and generate a dataset by running some concurrent codes that create contention.”. This is a source of internal threats to validity. Please mention this in the discussion section.

14. Page 32 (Section 3.3.1): It was mentioned that “A Java exemplary code emulating lock contention is executed in a controlled environment leveraging”. Is this your own code? if not, what is the source of the lock contention-causing codes?

15. Page 51: You mentioned that the perf command "perf-record" collects lots of symbols that are **not** related to contention faults, and then you mentioned that “a group of symbols appear with a high number of samples in case the code experience bad contention and with fewer samples when it experiences minimal contention”. This is a bit confusing, can you clarify this argument please?

16. Page 51: following the same discussion, you also mentioned that "Leveraging these symbols might help us to identify the contention fault types". How? if they are originally not related to contention faults?

17. Page 37: In comparison between figure 3.4 and 3.5 which illustrated "perf" snapshots when there's high contention, and low contention, respectively. What is the threshold that is used to decide if there a high vs. low contention?

18. Page 38: In the algorithm that automates the steps for faster log generation, in particular, in step 1 of the algorithm where you set the number of threads and sleep time. What is the model you followed to vary the numbers of threads and sleep times? Is it based on something from the literature?
19. Page 47: You have already mentioned that you used the DBSCAN as another clustering algorithm (in addition to the KMeans), and that the DBSCAN failed to produce the desired clusters out of the dataset. Have you tired other clustering algorithms such as, for example, Mini-Batch K-Means, Mean Shift, OPTICS, etc.?

20. Page 63: It was mentioned that “ JLM metrics do change based on the fault types, and once the metrics related to a particular fault are affected by that fault type, some impacts are

observed on the other metrics at the same time”. This is a very interesting insight! have any one in the literature reached to this conclusion as well?

21. Page 64: You had an interesting insight which is: “When the two faults occur at the same time, the metrics related to spin count increase in number **only** when the threads spend shorter period of time inside the critical section.”. But what if the threads spend longer time? what will happen in this case?

22. It would be good to mention the several threats to validity that must be assessed to better characterize the limitations of the approach and results presented in this thesis. The candidate can refer to Perry et al.’s paper (Empirical studies of software engineering: a roadmap. In *Proceedings of the conference on The future of Software engineering),* where the most relevant categories of threats to validity for this work are mentioned, such as: construct validity, internal validity and external validity.

Minor Comments:

1. Previous related work should be mentioned using the simple past tense.
2. Place your figures closer to the discussion that referenced them.
3. I suggest adding a table at the summary section of the literature review chapter, where the candidate summarized the related approaches, and the gaps.