Journal of Computer Science and Technology

Completed Review

Reviewer 2:

Reviewer Affiliation	Alibaba Group
Manuscript ID:	
Manuscript Type:	Original Article
Keywords:	Misconfiguration , Workload-related , Static analysis , Runtime monitoring
Speciality:	Software Systems
Date Submitted:	*blinded*
Manuscript Title:	
Date Assigned:	24-Apr-2022
Date Review Returned:	21-May-2022

req **Would you be willing to be recommended to other journal(s) of this field as a reviewer?**

Yes

Overview Report	**Outstanding**	**Good**	**Fair**	**Poor**	**Very Poor**
Content					
req Is the work relevant?			✓		
req Is the work original?		✓			
req Are the methods/proofs/experiments/etc. sound and convincing?				✓	
Presentation					
req Is the abstract an adequate summary of the work?		✓			
req Are the background and related work(s) clearly introduced?			✓		
req Are the methods/proofs/experiments/etc. properly stated?		✓			
req Are the conclusions clear and adequate?		✓			
req Are the references adequate?					

✓

req Is the presentation clear to the relevant audience? ✓

req Are the overall organization and length of the manuscript adequate? ✓

req Is the English satisfactory? ✓

Summary Report

req **Quality**

Not significant

Priority of Publication

If accepted, should the manuscript be prioritized for publication?

Accepted as

req **Confidence of your evaluation**

3(Confident)

req **Recommendation**

- Accept without revision
- Accept with minor revision
- Review again after major revision
- ✓ Resubmit after major revision
- Submit to another journal
- Reject

req **Would you be willing to review a revision of this manuscript?**

- ✓ Yes
- No

Comments

Confidential Comments to the Editor

Comments to the Author

This paper tackles a practically relevant and challenging problem:
preventing misconfigurations in production environment.
In particular, it focuses on the workload-related misconfigurations,
whose constraints can only be known at runtime.

To solve this problem, this paper conducts empirical study on four
popular open-source software systems and summarizes five code patterns
that are relevant to numerical configuration parameters.
It then proposes WMWatcher, which can analyze the target software's
source code, identify certain code patterns, and generate probes for
"branch interactions" with configuration parameters.
At runtime, WMWatcher utilizes these probes to generate runtime statistics
for system admins.
Last but not least, WMWatcher is implemented and evaluated on real
open-source software systems, with reasonable extra runtime overhead.

I recommend "Resubmit after major revision" for this paper due to the
following reasons.
- This paper over-claims its contribution: there is no evidence showing
that "WMWatcher could infer the constraints of configuration parameters"
- This paper fails to justify its significance compared with related work.

====== Over-claim of contribution =====

The introduction lists as contribution that "WMWatcher could infer the
constraints of configuration parameters under current workload for system
admins" (Page 3).
This is supposed to be a major contribution of this paper, as well as
a major novelty against existing work. I was curious to learn the rationale
behind the computation of such constraints.
However, later sections suggest that there is no automatic inference happening.
WMWatcher stops at the monitoring phase, and the actual constraints is
manually infered by system admins.
For example, none of the four evaluation cases in Sec. 4.4 shows the output
configuration constraints.
Such over-claim of contribution significantly weakens this paper.

Another inconsistency is in the introduction.
This paper uses MySQL's "thread_stack" parameter as a motivating example.
However, it is not shown throughout this paper whether WMWatcher is able to
prevent such misconfiguration.
In fact, it requires certain domain knowledge to relate the "thread_stack"
configuration with the runtime resource usage of MySQL.
This is beyond the capability of WMWatcher, which aims to identify and probe
configuration-related variables defined in the software source code.
As such, the motivating example indeed raises a challenging issue, but
this paper does not seem to solve it.

A good way of improving this paper is to clearly define its scope in the revised
version, properly position your work among prior work, and provide a motivating
example that matches WMWatcher's workflow.

===== Significance compared with related work =====

This paper aims to tackle misconfiguration prevention and compares with
existing work in this field.
However, a major challenge with preventing misconfigurations is how to compute
sound constraints that can benefit non-expert users.
Existing work, no matter based on static [17] or dynamic [24] techniques,
follow the setting as described above.

As a comparison, this paper tackles misconfiguration in a different setting.
Unlike previous work, it implicitly assumes that users possess sufficient
domain knowledge of the open-source software they are using, such that
the proposed WMWatcher outputs status monitoring instead of configuration
constraints directly.

This is a significant difference with related work, but is just skimmed over in the current writing, which may lead to misunderstanding.
This paper could be improved by thoroughly explaining the difference in its assumptions, and by justifying this assumption / properly limiting the scope of this paper.

Aside from the potential misunderstanding, a more severe issue in the current writing is that this paper fails to compare with related work on monitoring, or performance/parameter tuning. It would be much easier for readers to understand this work if this paper clarifies how it innovates in the status monitoring mechanism, or whether that is a contribution at all.

===== Minor comments =====

Sec. 2.2 "Handling Types After Interaction" is the major contribution of the empirical study, which further motivates the whole WMWatcher work.
The current writing explains well how each pattern could be recognized.
But this only provides engineering detail, not design insight.
This paper could be further improved by explaining the semantics implications of each syntactical pattern, as well as what kind of tradeoffs the user may face in each category.

Sec. 3.5, "monitoring status", should have been a highlight of this paper, since it is the final step of misconfiguration "prevention".
Yet, the current writing is not convincing.
For example, for the "Hard limit" pattern, WMWatcher records the v_vs that is mostly close to v_c (Sec. 3.5). How does remembering a single value help admins make the tradeoff, e.g. between efficiency and availability?
Further, how does this seemingly trivial metric compare with existing work on performance tuning?

Reviewer opted in to receive recognition on Publons? (Yes/No answer required)

- [ ] Yes
- [x] No

Use the below rating options to rate the reviewer on <u>this</u> submitted review. The rating options have corresponding numerical values which are averaged to determine an "R-Score" for reviewers. The "R-Score" for a reviewer displays as part of the reviewer search results to give you an indication of past performance.

Timeliness

- (●) Review was on time (Rating 3.0)
- ( ) Review was slightly delayed (Rating 2.0)
- ( ) Review was severely delayed (Rating 1.0)

Quality Assessment