Dear SIGMOD 2021 reviewers and area chairs

Thanks for your valuable and constructive comments on our submission titled “Optimizing recursive datalog evaluation based on ASLOG index”. In this author response, we try our best to address your questions and suggestions (listed as “ox” or detailed comments in your review). The detailed response to each of your comments is as follows.

Response to reviewer #3:

#3-o1: “It would be useful if the authors introduce subsections. For example in Sec 1, the authors describe three challenges but refer them later in the text through enumerations. Therefore, creating a subsection for the challenges and the contribution and clearly enumerating them would enhance the readability of the paper. ”

Thanks for the suggestion. If we get the chance to revise our submission, we would introduce subsection in section 1 as well as other sections to better improve the readability of the paper.

#3-o2: “While this may not be the main focus of the paper, a brief discussion of concurrency of the performance ASLOG would help readers get a perspective of behavior of the index. For example, the experimental setup deals with 4 cores- is that a conscious choice? Does increasing the #cores impact the sequence of query processing within ASLOG?”

Thanks for the comments and suggestion. It is true that the focus of ASLOG is non-parallel evaluation of recursive datalog rules, not concurrent evaluation. In our evaluation, we run the ASLOG-related experiments in a non-parallel setting only. The concurrent experiments were conducted with RDFox, a parallel baseline approach. We did try some different concurrency settings for RDFox, i.e., 2-cores, 4-cores and 6-cores. The results show that ASLOG outperforms 2-cores and 4-cores RDFox. When considering 6-cores RDFox, ASLOG outperforms it in calculating transitive closure (RQ2) and materializing increments (RQ3). In evaluating recursive rules (RQ1), 6-cores RDFox performs better than ASLOG in querying queries Q1, Q21, and Q22, ASLOG performs better for queries Q23 and Q24. As a result, we choose 4-cores RDFox as the compared baseline in the submission.

In our future work, we are planning to design and implement a parallel-version ASLOG. Currently, we have introduced parallel-based techniques in accelerating ASLOG’s searching in its TSI sub-index. Some primitive results show that such parallel searching could make ASLOG 1.26 times faster in a 4-cores setting. And we are working along this line.

#3-o3: ”A key observation that comes out from the experimental results (e.g. Fig 4, 5, and 6) is that there is a tradeoff space between ASLOG and other systems based on different query types and datasets. It would make sense if the authors capture that space and provide a depiction of which system works the best in different subareas of that space. This will enable readers to grow conventional wisdom and learn about the specific use cases and scenarios of datalog indexes in general such that given a query type one can now choose which system

to use."

Thanks for the suggestions. Due to the limitation of space, we failed to provide a more in-depth analysis on the experimental results in our submission. Some supplementary findings are listed as follows.

(i) ASLOG favors sparse datasets. As we explained in the submission that if we regard a dataset as a graph, COMP_TC slightly outperformances ASLOG in dense graph (i.e., dataset Gc2) when calculating transitive closures.

(ii) ASLOG favors larger datasets when ASLOG need to build index. For the smaller datasets, RDFox performs better than ASLOG, since the latter spends average 65% of its total time cost in building the index structure. For the larger datasets, ASLOG performs better than RDFOX, with the decreasing of the ratio that building index takes up. Nevertheless, when consider the computation time only (i.e., the index structure is built in advance), ASLOG outperformance RDFox in almost all datasets.

(iii) ASLOG favors smaller datasets when its index has been built in advance. When compare the computational cost of ASLOG and RDFox, we find that ASLOG's leading advantages decrease with the growth of the dataset's size. For the smaller datasets, the average speed up ratio of ASLOG is 139 times, and for the larger datasets, the ratio is 8.13 times. This shows that advantages of RDFox's parallel techniques.

(iv) Considering the types of queries, ASLOG is particularly suitable for the non-trivial applications of recursive rules, including calculating transitive closures, materializing increments, and querying queries with intersection operations. This echoes the third challenge we discussed in section 1 that it is important for an index structure to support various applications of recursive rules.

#3-o4: "Table 1 shows the overhead analysis of ASLOG. However, it would make sense to include the numbers from other systems. This is probably more important for the space analysis and not much for the time as we get a good picture of the performance of ASLOG (in terms of latency) in the previous subsections."

Thanks for the comments. Due to the limitation of space, we did not present the comparisons of space and time overhead of ASLOG with the baseline index structure, namely FI and AMEM. Due to its more complicated index structure, ASLOG's space and time overhead are more than FI's and AMEM's. However, we believe that such overhead is acceptable considering the advantages brought by ASLOG in evaluating recursive rules.

(i) Space overhead. Comparing with FI, ASLOG's space overhead is averagely 1.57 times larger (1.19 – 1.83). Comparing with AMEM, ASLOG's space overhead is averagely 1.52 times larger (1.10 – 1.86).

(ii) Time overhead. Comparing with FI, ASLOG's space overhead is averagely 1.55 times larger (1.04 – 2.74). Comparing with AMEM, ASLOG's space overhead is averagely 1.61 times larger (1.02 – 3.00).

Response to reviewer #4

#4-o1: “The supported syntax is not formally defined. It is not clear, for example, if a recursive rule can have a filter. If such rules are supported, how the evaluation is affected?”

Thanks for the question and comments. Due to the limitation of space, we did not present the full syntax of datalog supported by ASLOG in our submission. We give a BNF-based definition as follow.

Rules := head :- body
predicate := ordinary_predicate | not ordinary_predicate | built_in_predicate
head := ordinary_predicate (argument_list)
body : = basic_body | complex_body
basic_body := ordinary_predicate | ordinary_predicate, basic_body
complex_body := basic_body, built_in_predicate | basic_body, not ordinary_predicate
ordinary_predicate := string(argument_list)
built_in_predicate := variable op variable | variable op constant
op := < | > | =< | >= | = | !=

Also due to the limitation of space, we did not make it clear that ASLOG could support a datalog rule with a filter. Basically, it depends on the built_in predicate’s parameters. If a built_in predicate contains a constant, then the filter plays a filtering role when building AI index. More specifically, only the tuples that satisfying the built_in predicate (which can be determined when given the tuple’s arguments) are inserted into AI. If a built_in predicate contains two variables, then the filter works when establishing TSI. More specifically, only the transition tuples that satisfy the built_in predicate (which can be determined when given the transition tuple’s arguments) would be used to create an index entry in TSI index.

#4-o2: “I found section 2.2 hard to follow. I believe adding examples can help and improve this part.”

Thanks for the suggestion. We are sorry that we failed to make the concept of connection tuples, simple transition tuples, and forward/backward transition tuples clear in section 2.2. Please consider the following example. Let the EDB relation PAR(x, y) be as follows.

PAR(x, y)={t(0, 4), t(1, 2), t(7, 0), t(7, 3), t(2, 6), t(9, 0), t(4, 8), t(4, 5), t(0, 2)}

Then the connection tuples of t(7, 0) are t(0,2), t(0,4). The simple transition tuples of t(7, 0) are t(7, 2) and t(7 ,4). The forward transition tuples of t(7, 0) are t(7, 2), t(7, 6), t(7, 4), t(7, 8) and t(7, 5). There is no backward transition tuple of t(7, 0).

#4-o3: “The same for section 5.3. In the Connectivity of value and tuple paragraph ConV is used but never defined.”

Thanks for the comments. We are sorry that we did not give a detailed description of function ConV( ) in our submission. Similar to function Con(), we use ConV() to determine whether a tuple and a value are connected following a specific direction. The direction is determined by

the tuple's parameters. More specifically, if ConV(t, v)=TRUE, tuple t's second parameter is equals to v. Consider the example presented in the response to #4-o2. ConV(t(0, 4), 4)=TRUE, and ConV(4, t(0, 4))=FALSE; similarly, ConV(0, t(0, 4))=TRUE, and ConV(t(0, 4), 0)=FALSE.

#4-o4: "The definition of the Con is not clear. Why Con(t, t')= TRUE implies that Con(t', t)= FALSE. If, for example p i(x,y ) \in P i and also p i(y, x) \in P_i? Also, the use of t, t', t(x, y), t'(y,z)is very confusing. In my understanding, if t(x, y ) \in P i and t'(y, z) \in P_i. Then, t and t' are used without their arguments Maybe I'm not familiar with this notation, but I think it would be clearer to use t=p i(x,y) and t'=p_i(y, z)"

Thanks for the comment. We are sorry that we did not present the concept of function Con() very clear in our submission. Con (t, t ') is used to determine whether two tuples t and t' are connection tuples. Considering the example presented in our response to #4-o2, t'(0, 2) is the connection tuple of t(7, 0), so Con(t(7, 0), t'(0, 2))=TRUE. But Con(t'(0, 2), t(7, 0))=FALSE, since t(7, 0) is not a connection tuple of t'(0, 2). The only case that the above description would fail is that two reflexive tuples are connected to each other, i.e. t (x, x), t '(x, x) \in Pi. However, such case could be extremely rare in real datalog application, and it can be excluded by preprocessing the EDB relation.

Thanks for your constructive suggestion that present all tuples along with their parameters. We will modify the presentation of our submission accordingly. It could greatly improve the readability of the submission and help readers to better understand the introduced concepts.

#4-o5: "It would be interesting to see a theoretical complexity analysis of the proposed structure."

Thanks for the comment. Due to the limitation of space, we did not present the complexity analysis of the proposed ASLOG in our submission.
Let n be the number of tuples and m be the number of associated semantic index entries in each B+ leaf node.
The time complexity of building ASLOG index equals the sum of BT1 and BT2, where BT1 is the time complexity of building AI, and BT2 is the time complexity of building SI. We can derive that $BT1=n*\log_2(n)$, $BT2=m*n$. Therefore, the time complexity of building ASLOG index is $O(n*(\log_2(n)+m))$.
The time complexity of searching ASLOG equals to the sum of ST1, which is to search AI, ST2, which is to search CTI, and ST3, which is to search TSI. We can derive that $ST1=O(\log_2(n))$, $ST2=O(1)$, $ST3=O(m*\log_m(n))$. Therefore, the time complexity of searching ASLOG is $O(\log_2(n)+m*\log_m(n))$.

#4-o6-(1): "While the evaluation results presented in the paper seem promising, they are incomplete and could be improved:-What is the difference between ASLOGc and ASLOGm*?"

Thanks for the question. We are sorry that we did not explain the difference between ASLOGc and ASLOGm* clear in our submission.

ASLOGc means that only the time of calculation is counted, which includes the time of searching ASLOG, and the time of storing the result in memory. We use ASLOGc to compare the efficiency of ASLOG with RDFox, BTC and COMP_TC, in the scenarios that the ASLOG's index structure has been resided in the memory. In these scenarios, ASLOG can uses the index that resides in the memory to accomplish the evaluation of recursive rule.

ASLOG*m includes the time costs of parsing the rules, reading the index, searching ASLOG and storing the results in memory. The difference between ASLOG*m and ASLOGc is that the former has to process the rule and load the index from disk to memory. We also use ASLOG*m to compare the efficiency of ASLOG with RDFox, BTC and COMP_TC, in the scenarios that the index has been built and stored on disk in advance.

#4-o6-(2): "Can a query be computed without reading the index? "

Thanks for the question. Sorry for that we did not explain the meaning of "reading the index" clear in our submission. By "reading the index", we mean that load a pre-built index from disk to memory. As such, ASLOG could evaluate a query without reading the index if the index has already resided in the memory (the time cost of which is measured by ASLOGc). If the pre-built index is stored on the disk, then ASLOG has to read the index, and the time cost of evaluating rules is measured by ASLOG*m. Notice that in both cases, ASLOG uses pre-built index. For the scenarios that no index has been built, we use metrics ASLOGm and ASLOGd to measure ASLOG's efficiency.

#4-o6-(3): "The results presented do not give a full picture since many possible evaluations of the proposed algorithms and datasets are not presented (e.g, Ss1-Ss4 are not presented in figures 3, 4, 5, 6, 7 and the others do not include Sl1-Sl4 in $ASLOG_m$ is used in Fig 6a-6e, and ASLOGc is used in Fig 6f-6i). Either present all possible results or explain why some are missing/not applicable."

Thanks for the comments. Due to the limitation of space, we did not present the full experimental result in our submission. And we are sorry that we did not explain that clearly in our submission. For the results presented in Fig. 3-7, we also conducted the experiments on datasets Ss1- Ss4. Take Figure3 as an example, ASLOGd's average speedup ratio in evaluating query Q1 is 192.73 times, comparing with DLV, IRIS, XSB and Yap, and that ratio for ASLOG*d is 277.43 times. For query Q21, the average speedup ratio of ASLOGd is 270.59 times, and the ratio of ASLOG*d is 545.27 times. The case is similar for the result presented in Figures6(a)-(i), in which we did not present the result on dataset Gc1-Gc4 and Ss1-Ss4.

#4-o6-(4): "The evaluation is limited to a single program, with no negation or complex rules. It would be more compelling to see the results on other datalog programs. For instance, in [x] there is a benchmark (used in a different context) of datalog programs that could be used to evaluate ASLOG."

Thanks for the comments and suggestion. We are sorry that we did not we did not conducted

experiments on more complicated datalog rule in our submission. Since the focus of this submission is recursive rules only, we focus on a well-known and typical recursive datalog program only. However, ASLOG could support non-recursive rule and negation rules, since we have designed specific types of semantic index in its index structure. We did not conduct and present experiments on these types of rules due to the limitation of space. It is a great suggestion that to run our ASLOG on some third-party benchmarks. We will take it as a future direction of our work.

#4-o6-(4): "-Overhead of ASLOG: what was the baseline method? (overhead compare to what?)"

Thanks for the question. We are sorry that we did not compare the overhead of ASLOG to any baseline approaches in our submission. We have conducted experiments in comparing the overhead of ASLOG with other two index structures, namely FI and AMEM. However, for the sake of space, we did not present the result in our submission. Due to its more complicated index structure, ASLOG's space and time overhead are more than FI's and AMEM's. However, we believe that such overhead is acceptable considering the advantages brought by ASLOG in evaluating recursive rules.

(1) Space overhead. Comparing with FI, ASLOG's space overhead is averagely 1.57 times larger (1.19 – 1.83). Comparing with AMEM, ASLOG's space overhead is averagely 1.52 times larger (1.10 – 1.86).
(2) Time overhead. Comparing with FI, ASLOG's space overhead is averagely 1.55 times larger (1.04 – 2.74). Comparing with AMEM, ASLOG's space overhead is averagely 1.61 times larger (1.02 – 3.00)

#4-o7-(1): "Result presentation:-It would be better to include the queries used (written in datalog)."
"
Thanks for the suggestion. Due to the limitation of space, we did not list the included queries in our submission. The queries are listed as follows.
Q1:
Q21:
Q22:
Q23:
Q24:
Q3:
Q4:

#4-o7-(2): "-It is not clear why line charts are used and not bar charts in some of the figures, as they do not show any trends."

Thanks for the comment. We are sorry that we did not explain our legend very clear in our submission. We use different line charts or bar charts in different Figures because of the scale of ordinate. We use line charts for experimental data with large difference; otherwise, we use

bar charts. If we get the chance to revise our submission, we will change all bar charts to line charts.

#4-o7-(3): “Figure 4: RD Fox should be REFox n?
-Figure 5: for better presentation use consistent colors (possibly with different pattens).”

Thanks for the suggestions. RDfox in Figure 4 should be RDFoxn. Figures 5(a) and (b) should be changed to the same color and line charts.

#4-s1: “Datalog has been shown useful in various application domains, thus efficient evaluation of datalog rules is important.”

Thanks for the suggestion. We are sorry that we did not highlight the important of efficient evaluation of datalog rules in our submission. If we get the chance to revise our submission, we are planning to add more arguments and reference to show the importance of the reach question of the submission.

Response to reviewer #6

#6-o1-(1): “O1. The writing does not do justice to the results in this paper. See detailed comments below.
1. Refer to the first page, to rules R1 and R2. It is mentioned that both are recursive. But R1 is not recursive.”

Thanks for the comments and suggestion. We are sorry that we falsely describe both R1 and R2 as recursive rules in our submission. It should be revised to “…Rule R2 is a recursive datalog rule…”

#6-o1-(2): “Second page, paragraph starting with "Last but not least. "It is mentioned that PAR is evolving. But PAR is EDB, so it is fixed.”

Thanks for the comment. We are sorry that we failed to explain the situation in which PAR relation would be evolving very clear in our submission. In some emerging applications, especially in pervasive environment, EDB relations will be updated frequently (insert new tuples) according to the changes of the environment. Such update would potentially lead to the activation of the related datalog rules, which is referred to as “increments materializing” in our submission. During the evaluating of a rule, the EDB relation will not be changed.

#6-o1-(3): “The introduction and section 2. 1 is not well written at all. I would suggest to think harder on the presentation and write a totally different one. The challenges in section 2. 1 are not well presented: probably not needed at all or explain all on an example. The terminology in these sections is better to be thought over and changed to Datalog database terminology.”

Thanks for the comments and suggestions. We are sorry that we did not present section 1

and 2.1 very clear in our submission. The concept of constraints and constraint variables in Section 2.1 are the basis for building ASLOG index of datalog rule. On the one hand, the connectivity and transitivity of tuples can be easily described by these two terminologies. Based on them, the transitive relationship between EDB and IDB relations in the recursive rule body can be transformed into the transitive relationship between the tuples in EDB relation, which makes it possible to build the index for recursive rules. On the other hand, based on these two terminologies, we can describe the relationship between the predicate's arguments in the rule body, which provides convenience for automatically building the index of datalog rules.
It is a great suggestion to explain these concepts with an example. If we get the chance to revise our submission, we are planning to rewrite section 1 and 2 with an example to help readers to better understand the motivation and challenge of the submission.

#6-o1-(4): "An example of improvement: top of page 3: "by the association and dependency. " It seems like are introducing some important terminology which is not true. It is only plain English. You can explain this idea in the example suggested above."

Thanks for the comments and suggestion. We are sorry that we did not explain the term of "association and dependency" well in our submission. Basically, we are use these two terms to explain the relationship between the predicates' arguments in a datalog rule. We incorrectly make these two terms italic in our submission.

However, thanks to your valuable comments and suggestion, we find that these two terms could be explained in the terminology of datalog.

For non-recursive rules, such as R3 in Section 2, predicates par and person have an "association" on variable y. This is the basis for us to build the index of non-recursive rules. y is a constraint variable of rule R3. Since the association reflects a join operation, we call it join constraint variable. When building ASLOG, the constraint semantics is reflected in the tuple values of relations PAR and PERSON. For example, let t(Tom, Jane) \in PAR and t1(Jane, 35, F) \in PERSON. Because they satisfy the join constraint of rule R3, we consider that there is an association between tuples t and t1. Only these tuples constitute index entries which are inserted into JSI.

For a recursive rule, its constraint is the transitive relationship between EDB and IDB predicates. Since index cannot be built on IDB relation, building the index between EDB and IDB relations is converted to building the index on EDB relation. In fact, the semantics of recursive rules is transformed into the dependencies between the tuples in EDB relation. Therefore, the "dependency" here refers to the "transition dependency" between the tuples in EDB relation, which satisfies the transition constraint.

#6-o1-(5): "Another example of a part needed to be rewritten: page 12: The sentence about seminaive evaluation. What you explain is basic fixpoint evaluation. Seminaive evaluation tries to consider only the newly evaluated tuples in IDB predicates."

Thanks for the comments. We are sorry that we did not explained the semi-naïve evaluation well in our submission. Because the seminaïve algorithm evaluates the recursive rules iteratively, each loop makes the IDB produce some new tuples. The termination condition is that no new tuple is generated in the IDB, that is, the IDB relation reaches a fixpoint.

#6-o2: “The terminology used to express the ideas is low level programming language terminology. The ideas here could be better (and in a simpler way) expressed using Datalog database terminology.”

Thanks for the comments and suggestions. We are sorry that we did not use a well-structured datalog terminology to represent the basic concept of our ASLOG. Our goal is to design a new index for datalog rules. For non-recursive rules and negative rules, they can be expressed in traditional datalog terminology. However, for recursive rules, we feel that we need to introduce some new terminologies to better express them. Previously, in the field of datalog, the constraint was used for the conditions that should be satisfied between the rules. We introduce it into predicates in rules. With the constraint variable and connection variable, it is easy to determine the attributes of building AI and SI indexes. As we explained in our response to #6-o1-(4), we find that the introduced concept would be explained in the terminology of datalog. If we get the chance to revise our submission, we are planning to work on this direction.