CSE 510 Phase 3 Report

Group Members

Pavan Chikkathimmegowda
Krima Doshi
Tanner Greenhagen
Jack Myers
Shubham Verma
Christian Seto

April 25, 2022

Abstract

In Phase 3 of the Database Management Systems Implementation project, we implemented a BP TripleJoin class that joins a BasicPattern on the left side with an RDF triple on the right side based on a Subject or Object node ID provided by the user. We also had to modify the Tuple class provided by mini-base and use it to implement BasicPattern Class which stores the confidence and node IDs(Subjects and Ob- jects) fields. The iterator and sort classes are also modified accordingly to work with BasicPattern Class.

We are using three join techniques, Nested Loop Join, Index Nested Loop Join, and Sorted Index Nested Loop Join. At the end of this phase, we were able to study and compare disk accesses related to each of these techniques.

Keywords: *Database Management System Implementation, DBMS, Resource Description Framework, RDF, Probabilistic RDF, pRDF, Quadruples, buffer manager, iterator, btree, stream, sort, heapfiles, joins, BasicPatterns,BasicPattern Iterator, nested loop join, index nested loop join, sorted index nested loop join, BasicPattern Sort, nested joins.*

1 Introduction

RDF is a data structure which is used to store data in a simplistic manner. It consists of a combination of Subject, Predicate and Object. It is commonly used in an object-oriented context to solve real world problems as they have human-friendly syntax, easy data integration and aggregation compatibility (using transitive rules), yes/no queries, etc.[4]

RDFs enable multiple entities to be interlinked with each other, integrated into a richly connected data structure and queried as one. It entails many join queries with multiple input streams from index scans which are often star-shaped or chained. In this phase, we will implement algorithms which will enable us to perform such joins on an extended RDF - Quadruples.

Relational data can be accurately expressed using RDF's graph model. RDF has two node entities (subject and object), which are connected by a edge, which is denoted by a predicate. In addition, it can be extended by a confidence value, which denotes how much confidence this RDF triple has (together known as quadruples). It can be used in a probabilistic scenario to query triples.

In addition to this, we will use a data type known as BasicPatterns for joining. BasicPatterns are data types that store the confidence value along with node IDs of Subject and Object of fixed length to simplify the join process. Since the edges between two nodes will remain constant, we only store the mapped nodes(entities) in our result.

1.1 Terminology

- **BasicPattern** : They are data structures that store the Confidence value along with the Node IDs for Subject and Object of fixed length.
- **Nested Loop Join** : Nested Loop Join joins the BasicPattern on the left side along with RDF triple on the right side using a naive approach through two nested loops based on Object or Subject similarity.
- **Index Join** : Index join uses the indexes to join the record ID of a BasicPattern with a record ID of a RDF triple based on the join predicate that defines the index.
- **Sort Index Join** : Sort Index join first sorts the BasicPattern on the left and RDF triple on the right, and it then applies Index Join based on either Subject or Object of both the sets.
- **RDF Triple** : A label is a class that defines an ID and a String value pair to uniquely identify an element in the database. It is used when storing such string elements and creating btrees for them.
- **Entities** : They are real world objects that are distinguishable from other objects. In the quadruple use case, entities are generally the Subjects and Objects for which the relationship is defined. They have a String label and hence, can be stored with the label storage and access structure.
- **Predicates** : Predicates in the quadruple define the relationship between the two given entities and the confidence of that relationship. It can also be defined as the property of the entities given. They, like entities, are a String attribute and can similarly use the Label structure.
- **Confidence** : It is the value of percentage of tuples that satisfy a rule and a condition which is that fact asserted by that quadruple.

1.2 Goal description

As a phase 3 of this project, we have to implement BasicPatterns data structure. The quadruples created in Phase 2 are used as RDF triples containing IDs of (Subject, Predicate and Object) along with their confidence value.
The goal is to implement a BP Triple Join class that joins the BasicPatterns on the left with RDF triples on the right side using three join techniques : Nested Loop Join, Index Join and Sort Index Join. Along

with this class, the iterator, sort and command line classes are also modified to work with BasicPatterns and their joins.

1.3 Assumptions

We make the following underlying assumptions for our database:

1. We assume that the user will always provide valid commandline input, with no errors or mistakes.
2. It is assumed that any input file will have a name without any spaces.
3. It is assumed that the subject, predicate, and object are all separated by colons. It is also assumed that the object and the confidence are separated by tabs.
4. The user knows the command input order and corresponding type in the command.
5. Any corrupt row is not entered into the database by the batchinsert command.
6. The confidence field is always returned.
7. The node ids start from 0, but is the node after the confidence.
8. The query files are initially accurate and can be modified based on our node numbering system to work for our query program.
9. When filtering confidence, we return all items with a confidence greater than or equal to the inputted confidence. The other filters are strictly based on equality.

2 Description of the Proposed Solution/ Implementation

This is the Phase 3 of CSE 510 : Database Management Systems Implementation project. We implemented BasicPatterns data structure that store the node IDs and contain (Confidence, Subject and Object) fields. Additionally, we implemented a BP Triple Join class that joins the BasicPatterns on the left side with RDF triples on the right side using three join techniques : Nested Loop Join, Index Join and Sort Index Join. Moreover, the iterator class, sort class and command line class are also modified to work with BasicPatterns and their joins.

The query tree for our implementation would filter first and then send the filtered rows for the join. After the joins, the result would be fed into the Sort.

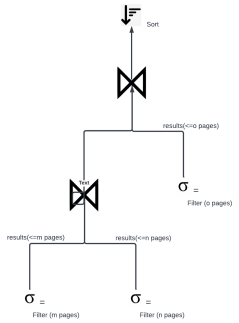

Figure 1: Sort-Join Query Tree

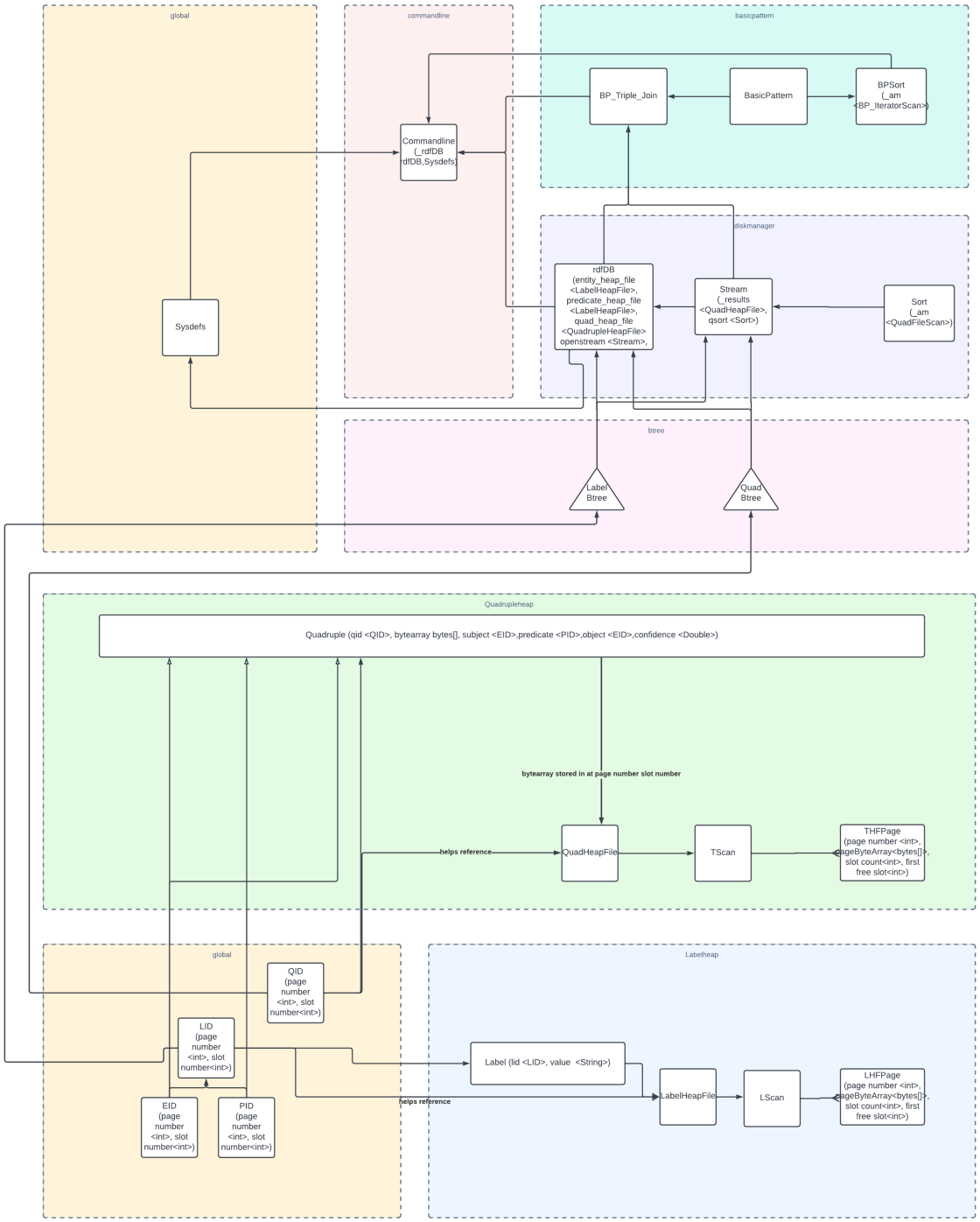

Figure 2: Architecture diagram for phase 3 (on top of architecture given for minibase[6])

2.1 BasicPattern

Since this phase is building on top of minibase, we chose to design the BasicPattern class to mirror a tuple structure. The thought was that this would make it easier to later implement other functionalities like writing to heap files and sorting as we could easily transform the BasicPatterns into tuples. We designed our BasicPattern class to have four main fields: data, length, offset, and field count. We set it up so the first two bytes of the data array would contain the number of fields and then the remaining information would be double and integer values. We stored the number of fields in the first two bytes to make sure that even if we just had the data array, we would still know how many fields the BasicPattern had.

At a high level, each BasicPattern consists of a double value for the confidence followed by an nodeID for each node it contains. More specifically, the nodeID is represented by an EID for each subject or object. The EID is comprised of two four-byte integer values, the first for the page number and the second for the slot number. These two values are used to retrieve the label from the entity label heap file that is stored in the rdfDB. Since each field in the BasicPattern is 8 bytes long, the length of the data array is two bytes (number of fields) plus eight bytes (confidence value is a double) plus eight bytes * number of EIDs. This is helpful when it comes to accessing the actual fields. Compared with the tuple class, in which string fields can have variable length and some fields have different fixed lengths (short and int fields are of different fixed lengths 2 and 4 bytes respectively), our BasicPattern class does not need a field offset array. We know that the starting byte of each field in the BasicPattern can simply be accessed from 2 + (field number - 1)*8. We choose for the field numbers to start at 1 to again mirror the tuples and provide some form of consistency.

The BasicPattern will always store the confidence in the first field. Since our BasicPatterns will always have a confidence field we decided to put it first and have every EID after the confidence field. This makes it easy to know what each field's type is without having to actually have that information stored anywhere. Besides a default constructor we also created a constructor that converted a Quadruple into a BasicPattern and a Tuple into a BasicPattern so that we could read QuadrupleHeapfiles and regular Heapfiles and instantly converting Quadruples and Tuples into BasicPatterns when iterating through those files. We also created a further constructor that takes in an existing BasicPattern and an array filled with the desired projected field numbers that we use to take an existing BasicPattern and apply projection. To allow for conversion between BasicPatterns and Tuples we had to additionally modify the tuple to allow insertion and retrieval of double values.

2.1.1 Design Decisions:

- Base the design of BasicPatterns on the existing minibase tuples, but with slight modifications. The first two bytes hold the number of fields in the BasicPattern. Each field after that has a fixed length of eight bytes. Hence, a field offset array is not needed. In order to calculate the offset of a field, the only information needed is the field number and the length of the data array.
- The first slot will always hold the confidence as a double. Every BasicPattern will always have a confidence, so by placing it in the first slot this makes it easy to set and get the confidence as needed. This way we will always know what type each field is storing without having to explicitly store that information.
- Each successive slot store EIDs in the BasicPattern to represent nodeIDs. Each node in the BasicPattern is represented as an entity in our database, and is thus stored in the entity Heapfile in in the rdfDB.
- Each entity has exactly one entry in the entity Heapfile. This allows for easy comparisons of nodes, without actually needing to check the string value of the node. We can simply compare two EIDs, and if they are the same then we know the nodes have the same value.
- BasicPatterns can be converted into Tuples, and vice versa. This allowed for us to leverage the existing Heapfile class to store the BasicPatterns rather than create a new BasicPattern Heapfile class.

2.2 BPIterator

The main purpose of our BPIterator is to read over QuadrupleHeapfiles and Heapfiles and change the Quadruples and Tuples. The iterator uses either a TScan or a Scan to go over the Quadruples and Tuples and the uses the BasicPattern Quadruple and Tuple constructors and returns a BasicPattern. It also can be used to see if there are any records within the file as when we are doing joins and sorting we are not going to proceed if the underlying files are actually empty.

2.3 BP_Triple_Join

BP_Triple_Join class is used to join RDF with itself using a BasicPattern and a RDF Quadruple. First, a BasicPatternIteratorScan is created from the filtered RDF heapfile corresponding to the input filters given for the left RDF. Next, this iterator is passed into the BP_Triple_Join class and all variables required for the join are set. After that, the join method is called corresponding to the input given.
Command used:

query <RDFDB> <query_file> <buffer_pages> <join_number>

where join_number can be:

1. Nested Loop Join: A Stream is created to filter the Heapfile containing the Quadruples. Then, we iterate over all the quadruples in the stream for the current basicpattern and check if the join condition matches. If a match is found, we add it to our result heapfile, else we go continue until all the quadruples in the RDF stream have been completed. After the inner loop completes, we go to the next BasicPattern until all the BasicPatterns are traversed.

2. Index Nested Loop Join: A Stream is created to filter the Heapfile containing the Quadruples with the given basicpattern join column value. Then, we iterate over all the quadruples in the stream and join it with the given basicpattern until all the BasicPatterns are traversed and add each to the resultant heapfile.[1]

3. Sorted Index Nested Loop Join: A sorted Stream is created to filter the Heapfile containing the Quadruples with the given basicpattern join column value. Then, we iterate over all the quadruples in the stream and join it with the given basicpattern. The next BasicPattern is then fetched and after that it is compared with the current BasicPattern. If they match, the current stream is re-scanned, else a new stream is created by filtering with the given filter values and join BasicPattern column value. The process continues until all the BasicPatterns are completed.

2.3.1 Design Decisions:

- The join types selected are nested loop join, index nested loop join, and sort index nested loop join.
- Nested loop join is the most basic and universal join that is used widely by most of the applications. It queries the data quadruple by quadruple. So it is the join that usually has the most number of disk accesses.
- Index nested loop join find the elements that match the join value using the index and joins the results. It is faster than the nested loop join and can be used whenever an index is available for your inner table for faster joins and less disk accesses.
- Sorted nested loop join is similar to the index nested loop join, the only difference being that we do not need to stream the inner RDF again for consecutive join column values which are the same, resulting in even less disk accesses.
- The resultant confidence in the joined row is the minimum of the confidence of the two rows that were involved in the join.

2.4 BPSort

A new class was created titled BPSort that was build directly off of the original tuple sorting Sort class. The sort runs by performing an initial heap sort and then proceeds by performing a merge sort of the BasicPatterns. We stuck with this implementation as it performed very well, required very little modification from the original Sort class, and reduces the number of IO operations needed as the initial runs are larger (assuming a large enough set of sorting buffer pages is set). The main design decision was to convert all of the BasicPatterns into Tuples so that the system would basically utilize the functionalities of the original minibase. This means that each BasicPattern would be converted into a Tuple. The Tuples would have 1 less than twice the number of fields as each EID field would be converted into two integer fields within the tuples (page number and slot number). This meant that when initializing the sort the sorting field had to be updated to the correct Tuple Sorting field meaning it has to be increased. If the sorting field was set to -1 the first field was used as it held the confidence.
Additionally the comparison method between the tuples also had to be changed to account for the fact that comparisons were being done on strings that were accessible by the page and slot integers and not the page and slot integers themselves. This required accessing the label heap file and grabbing the labels whenever tuples were compared by an integer value. An additional comparison for double values also had to be made that just directly compared the values, but since tuples weren't originally designed to deal with doubles the comparison had to be added.

2.5 CommandLine

The CommandLine class serves as the layer that allows for the user to run programs that act on the database. It is responsible for handling user input and parsing query files to get the necessary fields and then initiate the program. It is in charge of running and setting up the system to work with the query program. It checks that the entered command has the correct number of options associated with it and then begins the program. An error message is shown if the user provides invalid input. The batchinsert, query, and report programs are ran from this class. We will assume that no further explanation is required for the batchinsert and the report functionalities as they remain the same as before. Please reference 3 for details on the syntax of how to call each of the three programs.
The first act of the query program is to open the specified rdfDB and set the buffer pages via a call to the SystemDefs class. Then, the specified query file is loaded. The white space is removed and the contents of the query file are parsed using regular expressions. Once all of the required inputs are parsed from the query file, the join specified by the user is then loaded. A Stream of quadruples is then opened, which applies the specified subject, predicate, object, and confidence filters. The Stream is provided as input to create a BasicPatternIteratorScan for the left side. The left iterator and the other inputs from the query file are passed to a BP_Triple_Join object. The first join is ran, and then its outputs are loaded into a new BasicPatternIteratorScan. This iterator is passed as the left side of the second BP_Triple_Join, which will run the same join type as the first. Finally, a third BasicPatternIteratorScan is opened, and is provided as input to a BPSort object. The output of the BPSort is then printed to the terminal.

2.5.1 Design Decision:

- We assume the user to input the command in the correct format for it to work, otherwise it would throw an error and end the current run. We also assume the query file is in the expected proper format as the examples given by the professor on ED Discussions.
- The number of buffers entered by the user is sufficiently large to be able to execute the query.

3 Interface Specifications

To run the actual code outside of an IDE a jar must be created. In our case we used the Gradle build system (version included in System Requirements) to package our jar. If you navigate into the CSE510_Project directory, you can then run the command ./gradlew build. This will actually create the jar which can then

be found in the build/libs directory named CSE_510_Project.jar. To run the jar, assuming you are in the build/libs directory, can be done via the command java -jar CSE_510_Project.jar.

The jar executable takes three command line arguments (items in all capital letters need to be replaced with actual values when running the commands):

1. batchinsert DATAFILENAME INDEXOPTION RDFDBNAME

    a) The DATAFILENAME should be the original file holding the quadruples
    b) The RDFDBNAME will be the name of the new database. When this name is later referenced in query or report the name the user enters should be RDFDBNAME_INDEXOPTION
    c) The INDEXOPTION should be values from 1 through 5 with their meaning being outlined below:
        i. Object
        ii. Predicate
        iii. Subject
        iv. Object + Predicate
        v. Predicate + Subject

2. query RDFNAME QUERYFILE NUMBUF JOIN_TYPE

    a) The NUMBUF should be whole numbers and relatively large (100 or greater recommended, smaller could fail as runs can't fit during sorting)
    b) RDFNAME is the database name which is created by inserting data by previous phases of the project. RDFDBNAME_INDEXOPTION where RDFDBNAME and INDEXOPTION are from the batchinsert command.
    c) QUERYFILE is the text file where all query parameters are stored in below format
    d) The JOIN_TYPE for this are
        i. Enter a 1 for a Nested Loop Join
        ii. Enter a 2 for Index Nested Loop Join
        iii. Enter a 3 for Sorted Index Nested Loop Join

```
S( J(
  J([SF1,PF1,OF1,CF1],
  JNP,JONO,RSF,RPF,ROF,RCF,LONP,ORS,ORO
  ),
  JNP,JONO,RSF,RPF,ROF,RCF,LONP,ORS,ORO
  )
  SO, SNP, NP
  )
```

        iv. SF1 - Subject filter
        PF1 - Predicate filter
        OF1 - Object filter
        CF1 - Confidence filter
        JNP - BPJoinNodePosition
        JONO - JoinOnSubjectorObject
        RSF - RightSubjectFilter
        RPF - RightPredicateFilter
        ROF - RightObjectFilter
        RCF - RightConfidenceFilter
        LONP - LeftOutNodePositions

ORS - OutputRightSubject
ORO - OutputRightObject
SO - BP Sort Order
SNP - Sort NodeID Position
NP - Number of pages

3. report FILENAME

   a) FILENAME is the database name which is created by inserting data by previous phases of the project. RDFDBNAME_INDEXOPTION where RDFDBNAME and INDEXOPTION are from the batchinsert command.

4 System Requirements

4.1 Required Tools/Software

1. Java 8 or greater : Lower versions of Java is not compatible with Gradle which affects automation in the project.
2. JDK 1.8 or Greater : JDK version less than 1.8 would not support Java 8 or above which can cause the project to not run properly.
3. Gradle 7.4.1 : Helps in fetching required libraries and plugins for Java projects automatically. Gradle version 7.4.1 is compatible with Java 8 which is used in this project.
4. Minibase : Minibase folder provided to us in Phase 1 has been modified to work on quadruples. The tar file is unzipped and then opened in any of the IDEs. We have used VS Code for our build.
5. Linux : The project will work only in a linux system or in Windows with WSL (Windows Sub-system for Linux).

4.2 Execution Instructions

1. Go to the project home directory "CSE510_Project"
2. Run the command "gradle build". This will produce the artifact "build/libs/CSE_510_Project.jar"
3. Run the command "java -jar build/libs/CSE_510_Project.jar" to execute the application.
4. Enter the desired command from 3 for the application to perform.

5 Related Work

To increase the performance of RDF dataset, RDF need to be partitioned through various systems while querying the data. Although, the current technologies run flawlessly on a single node system, they become highly inefficient while running on multiple systems. As a solution to this problem, Huang's paper introduced a scalable RDF data management system which is three times more efficient than any other popular multi-node RDF data management system, by introducing the following techniques for (1) leveraging best single node RDF-store technology (2) data partitioning across nodes to accelerate query processing using locality optimizations and (3) SPARQL queries are decomposed into fragments having high performance leveraging data partitioned within in a cluster.[2]

Technologies leveraging Big Data like NoSQL DBMS ensures scalability and high availability of Web data. As the amount of data on web is increasing continuously, an intuitive technology is required which should be scalable and available to fully use the data and metadata at this level of storage. A lot of research efforts have already been made to implement a distributed RDF data management system. These systems

are built on NoSQL to ensure availability, scalability and high performance. Banane's paper represents a qualitative study of most commonly used existing systems dedicated to manage RDF data at large volumes, that are generally based on NoSQL database management systems. [5]

A single node system RDF data store faces faces a major issue of the lack of scalability. Increasing use of Cloud computing has introduced a way to create a distributed ecosystem of RDF triple stores that has inherently helped to introduce distributed query processing properties in turn increasing the scalability to support upto a planet scale storage. Khadilkar's paper introduces Jena-HBase which is a HBase backed triple store that is built on Jena framework. Jena-HBase provides an efficient and scalable storage along with effective querying solution which supports all the capabilities and properties from RDF specification. [3]

There has been a deep focus and research on RDF and related Semantic Web technologies, that introduced new specifications for RDF and OWL. However, most of the implementations are only efficient on a small scale single node system and RDF/ OWL implementations become inefficient when applied on a large-scale world of semantic web. In essence, the implementations of RDF which efficiently scale to a huge enterprise-class data sets are required. Jena2 is 2nd generation implementation of Jena which is a leading toolkit for semantic web programmers. Wilkinson's paper implements Jena2 subsystem that supports large datasets at a huge scale. Wilkinson et all described the features of Jena2, modifications compared to Jena1 and all the essential steps for the implementation and performance tuning issues. The paper also talks about the requirement of Query optimization for RDF which Wilkinson et all identified as a promising area for future research. [**effrdfstorage**]

6 Conclusions

For our analysis, we performed several join queries with the same filters on both the left and right side, as seen in figure 3.

Query	Subject	Predicate	Object	Confidence	Number of Instances
1	*	Image	*	*	4
2	Kyo_Nagashima	*	*	*	41
3	*	depiction	*	*	52
4	*	firstName	*	*	74
5	crschmidt	knows	*	*	78
6	crschmidt	*	*	*	96
7	*	weblog	*	*	151
8	Christopher_Richard_Gabriel	*	*	*	11
9	*	*	male	*	11
10	*	weblog	*	0.9	17
11	*	homepage	*	0.9	20
12	Schuyler_Erle	knows	*	*	25
13	*	schoolHomepage	*	*	11
14	*	weblog	*	0.8	35
15	*	knows	*	0.9	43
16	David_Dorward	*	*	*	48
17	*	workplaceHomepage	*	*	50
18	*	*	Person	0.9	45
19	*	aimChatID	*	*	51
20	*	name	*	0.9	44
21	*	*	Mr	*	57
22	*	seeAlso	*	0.8	59
23	*	seeAlso	*	0.75	67
24	*	knows	*	0.8	75
25	*	interest	*	*	89
26	*	knows	*	0.75	98
27	*	name	*	0.8	96
28	*	seeAlso	*	0.5	131

Figure 3: Table of queries used for analysis

Further, our analysis only focused on "self joins" (i.e., data that joins on itself). The purpose of this was to simplify our analysis of the time complexities of our join algorithms. Specifically, we performed regression analysis on the number of instances of each filter (see "Number of Instances" in figure 3) versus the number of reads and writes (summed together as total I/O) of a single join. We set our buffer size to be 100 frames and queried from the example RDF dataset from phase 2 with a sorting scheme of 1 (quadruples sorted on objects). The following are the results of each join.

6.1 Nested Loop Join

Theoretically, the time complexity of nested loop join is $N + N^2$ where N is the number of pages. Thus, we should see a quadratic relationship between the number of tuples in each filter. We performed a quadratic regression model where $y = \beta_0 + \beta_1 x + \beta_2 x^2$ with y as the total number of reads and writes, and x as the number of tuples/instances:

```
Quadratic Fit Statistics for Nested Loop Join
----------------------------------------------
[118656.82729268  39256.40181656    123.19998532]
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.867
Model:                            OLS   Adj. R-squared:                  0.856
Method:                 Least Squares   F-statistic:                     81.32
Date:                Mon, 25 Apr 2022   Prob (F-statistic):           1.14e-11
Time:                        17:54:43   Log-Likelihood:                -420.11
No. Observations:                  28   AIC:                             846.2
Df Residuals:                      25   BIC:                             850.2
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       1.187e+05   4.39e+05      0.270      0.789   -7.86e+05    1.02e+06
x1          3.926e+04   1.41e+04      2.791      0.010    1.03e+04    6.82e+04
x2           123.2000     96.169      1.281      0.212     -74.863     321.263
==============================================================================
Omnibus:                       25.079   Durbin-Watson:                   2.401
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               38.794
Skew:                          -2.103   Prob(JB):                     3.77e-09
Kurtosis:                       6.944   Cond. No.                     1.90e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.9e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
----------------------------------------------
```

Figure 4: Nested Loop Join full quadratic model

As the quadratic model in figure 4 suggests, the constant β_0 is not significant ($p - value = 0.789$, $-7.86e+85 \leq \beta_0 \leq 1.02e+06$) as well as the quadratic coefficient β_2 ($p - value = 0.212$, $-74.863 \leq \beta_2 \leq 321.263$). Meanwhile, β_1 is significant ($p - value = 0.010 < 0.05$, $1.03e+04 \leq \beta_1 \leq 6.82e+04$). Thus, the model should be rerun without the non-significant features. To be defensive, each of these will be removed one at a time, starting with the constant β_0.

As the revised quadratic model in figure 5 suggests, the quadratic coefficient β_2 is not significant ($p - value = 0.115$, $-27.040 \leq \beta_2 \leq 235.113$). Meanwhile, β_1 is still significant ($p - value = 0.000 < 0.05$, $2.91e+04 \leq \beta_1 \leq 5.61e+04$). Thus, the model should be rerun without β_2.

As the linear model suggests, β_1 is significant ($p - value = 0.000 < 0.05$, $4.77e+04 \leq \beta_1 \leq 5.75e+04$) with no other coefficients or constants to remove. With an adjusted R^2 of 0.945, this mode fits very well with the data. Unfortunately, this result does not align with out theoretical model of $N^2 + N$. This could be for several reasons. One is that the power of our experiment ($n = 28$) is not high enough to verify our results. Also, it could also be that our chosen range for the x-axis is too limited since we were limited by time to run our join queries. Given that the the quadratic coefficient β_2 had a p-value of 0.115 in the revised quadratic model and an adjusted R^2 value of 0.948, this does seem to suggest that the model has a quadratic component. Figure 7 contains the plot of the data with both the revised linear and quadratic models.

```
Quadratic Fit Statistics for Nested Loop Join (no intercept)
--------------------------------------------
                            OLS Regression Results
==============================================================================================
Dep. Variable:     Total I/O (1st Join)   R-squared (uncentered):                   0.951
Model:                              OLS   Adj. R-squared (uncentered):              0.948
Method:                   Least Squares   F-statistic:                              255.0
Date:                  Mon, 25 Apr 2022   Prob (F-statistic):                    8.22e-18
Time:                          20:13:55   Log-Likelihood:                         -420.15
No. Observations:                    28   AIC:                                      844.3
Df Residuals:                        26   BIC:                                      847.0
Df Model:                             2
Covariance Type:              nonrobust
==============================================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
Number of Instances   4.26e+04   6563.533      6.490      0.000    2.91e+04    5.61e+04
(Num. Instances)^2    104.0363     63.768      1.631      0.115     -27.040     235.113
==============================================================================
Omnibus:                       25.685   Durbin-Watson:                   2.396
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               41.129
Skew:                          -2.125   Prob(JB):                     1.17e-09
Kurtosis:                       7.146   Cond. No.                         289.
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Figure 5: Nested Loop Join quadratic model (no intercept)

```
Linear Fit Statistics for Nested Loop Join (no intercept)
--------------------------------------------
                            OLS Regression Results
==============================================================================================
Dep. Variable:     Total I/O (1st Join)   R-squared (uncentered):                   0.947
Model:                              OLS   Adj. R-squared (uncentered):              0.945
Method:                   Least Squares   F-statistic:                              478.0
Date:                  Mon, 25 Apr 2022   Prob (F-statistic):                    1.05e-18
Time:                          20:15:48   Log-Likelihood:                         -421.51
No. Observations:                    28   AIC:                                      845.0
Df Residuals:                        27   BIC:                                      846.4
Df Model:                             1
Covariance Type:              nonrobust
==============================================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
Number of Instances  5.261e+04   2406.315     21.862      0.000    4.77e+04    5.75e+04
==============================================================================
Omnibus:                       27.201   Durbin-Watson:                   2.175
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               45.941
Skew:                          -2.229   Prob(JB):                     1.06e-10
Kurtosis:                       7.417   Cond. No.                         1.00
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Figure 6: Nested Loop Join linear model (no intercept)

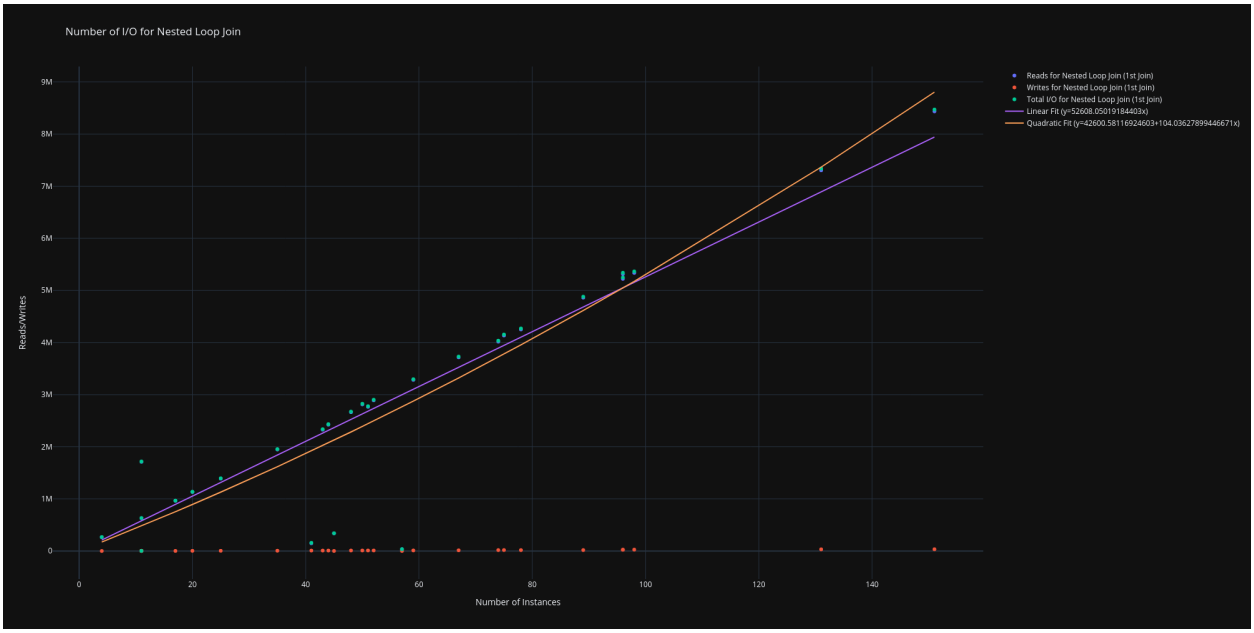

Figure 7: Plot of total reads/writes for nested loop join

6.2 Index Nested Loop Join

Theoretically, the time complexity of an index nested loop join is $N + (N \times p \times s) = N(1 + p \times s)$ where N is the number of pages, p is the average number of tuples per page, and s is the cost of searching. Thus, since we use the same index for each join (the entity b+ tree), cost of finding a matching tuple should be the same on average across all queries. Therefore, $(1 + p \times s)$ is constant, making the time complexity of an index nested loop join linear. Thus, we performed a linear regression model such that $y = \beta_0 + \beta_1 x$ with y as the total number of reads and writes, and x as the number of tuples/instances:

As the linear model in figure 8 suggests, β_0 is not significant ($p - value = 0.733$, $-121e + 06 \leq \beta_0 \leq 8.62e+05$) while β_1 is significant ($p - value = 0.000 < 0.05$, $3.12e+04 \leq \beta_1 \leq 6.22e+04$). Thus, the revised linear model should have the constant β_0 removed.

As the revised linear model in figure 9 suggests, β_1 is significant ($p - value = 0.000 < 0.05$, $3.63e + 04 \leq \beta_1 \leq 5.27e + 04$) with no other coefficients or constant to remove. Thus, with an adjusted R^2 of 0.816, the revised linear model seems to fit the data and supports the theoretical time complexity of index nested loop joins. Figure 10 contains the plot of the data with the revised linear model.

6.3 Sorted Index Nested Loop Joins

Theoretically, the time complexity of a sorted index nested loop join is should be similar to index nested loop join, which is $N + (N \times p \times s) = N(1 + p \times s)$ where N is the number of pages, p is the average number of tuples per page, and s is the cost of searching. Thus, $(1 + p \times s)$ is constant, making the time complexity of a sorted index nested loop join linear. However, since the index is sorted, the cost of searching should be lower, resulting in a smaller slope. We performed a linear regression model such that $y = \beta_0 + \beta_1 x$ with y as the total number of reads and writes, and x as the number of tuples/instances:

As the linear model in figure 11 suggests, β_1 is not significant ($p - value = 0.686$, $-299.607 \leq \beta_1 \leq 447.810$) while β_0 is significant ($p - value = 0.000 < 0.05$, $4.97e + 04 \leq \beta_0 \leq 9.75e + 04$). Thus, the revised linear model should have the coefficient β_1 removed.

As the new model in figure 12 suggests, β_0 is significant ($p - value = 0.000 < 0.05$, $6.46e + 04 \leq \beta_0 \leq 9.05e + 04$) with no coefficients or constants to remove. Thus, the revised model seems to suggest that the sorted nested loop join has a constant time complexity. However, this contradicts our theoretical model of a linear model. This could be caused by a couple of issues. One is that the buffer frames is too large, cause a

```
Linear Fit Statistics for Index Nested Loop Join
------------------------------------------------
[-173652.41411564   46689.86864803]
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.595
Model:                            OLS   Adj. R-squared:                  0.580
Method:                 Least Squares   F-statistic:                     38.26
Date:                Mon, 25 Apr 2022   Prob (F-statistic):           1.53e-06
Time:                        17:54:43   Log-Likelihood:                -435.50
No. Observations:                  28   AIC:                             875.0
Df Residuals:                      26   BIC:                             877.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -1.737e+05   5.04e+05     -0.345      0.733   -1.21e+06    8.62e+05
x1          4.669e+04   7548.089      6.186      0.000    3.12e+04    6.22e+04
==============================================================================
Omnibus:                       19.470   Durbin-Watson:                   2.423
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               23.138
Skew:                          -1.855   Prob(JB):                     9.46e-06
Kurtosis:                       5.463   Cond. No.                         125.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
------------------------------------------------
```

Figure 8: Index Nested Loop Join linear model

```
Linear Fit Statistics for Index Nested Loop Join (no intercept)
---------------------------------------------------
                                 OLS Regression Results
=======================================================================================
Dep. Variable:     Total I/O (1st Join)   R-squared (uncentered):                   0.823
Model:                              OLS   Adj. R-squared (uncentered):              0.816
Method:                   Least Squares   F-statistic:                              125.3
Date:                  Mon, 25 Apr 2022   Prob (F-statistic):                    1.20e-11
Time:                          21:13:35   Log-Likelihood:                         -435.57
No. Observations:                    28   AIC:                                      873.1
Df Residuals:                        27   BIC:                                      874.5
Df Model:                             1
Covariance Type:              nonrobust
==========================================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------
Number of Instances  4.449e+04   3975.335     11.192      0.000    3.63e+04    5.27e+04
==============================================================================
Omnibus:                       17.533   Durbin-Watson:                   2.411
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               19.388
Skew:                          -1.738   Prob(JB):                     6.17e-05
Kurtosis:                       5.129   Cond. No.                         1.00
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Figure 9: Index Nested Loop Join linear model (no intercept)

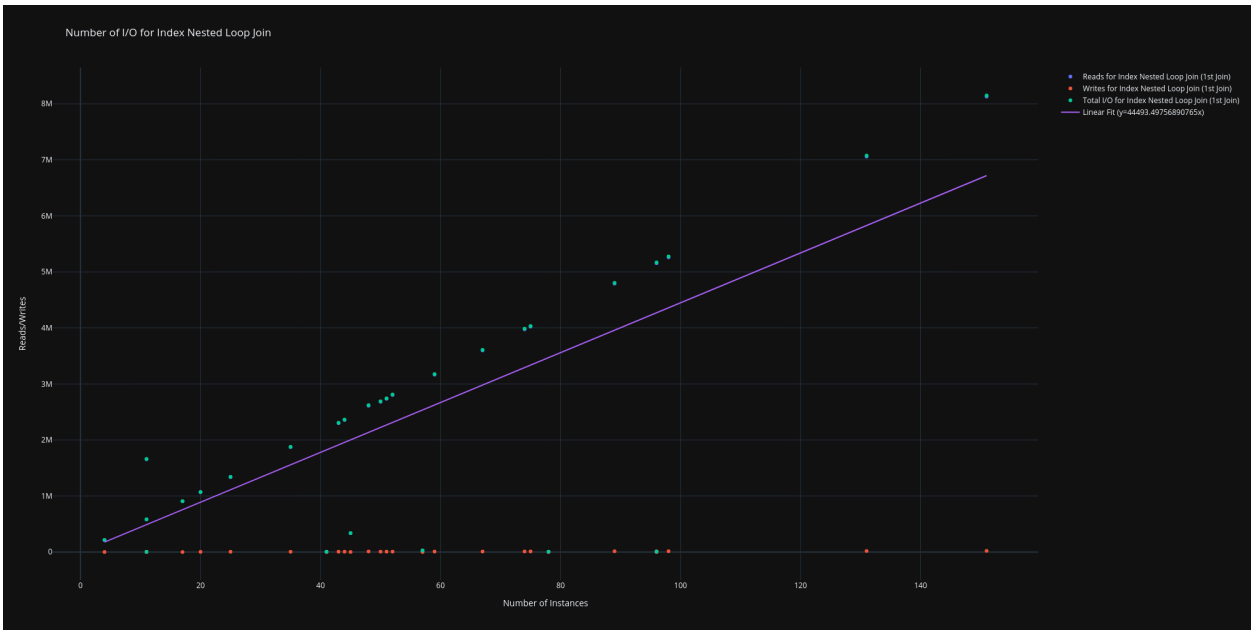

Figure 10: Plot of total reads/writes for index nested loop join

```
Linear Fit Statistics for Sorted Index Nested Loop Join
------------------------------------------------
[73589.09016272    74.10141007]
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.007
Model:                            OLS   Adj. R-squared:                 -0.034
Method:                 Least Squares   F-statistic:                    0.1675
Date:                Mon, 25 Apr 2022   Prob (F-statistic):              0.686
Time:                        17:54:43   Log-Likelihood:                -306.05
No. Observations:                  26   AIC:                             616.1
Df Residuals:                      24   BIC:                             618.6
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       7.359e+04   1.16e+04      6.353      0.000    4.97e+04    9.75e+04
x1            74.1014    181.069      0.409      0.686    -299.607     447.810
==============================================================================
Omnibus:                       17.593   Durbin-Watson:                   2.385
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               18.990
Skew:                          -1.910   Prob(JB):                     7.52e-05
Kurtosis:                       4.716   Cond. No.                         116.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
------------------------------------------------
```

Figure 11: Sorted Index Nested Loop Join linear model (no intercept)

```
Linear Fit Statistics for Sorted Index Nested Loop Join (no intercept)
-----------------------------------------------
                            OLS Regression Results
==============================================================================
Dep. Variable:     Total I/O (1st Join)   R-squared:                      -0.000
Model:                              OLS   Adj. R-squared:                 -0.000
Method:                   Least Squares   F-statistic:                       nan
Date:                  Mon, 25 Apr 2022   Prob (F-statistic):                nan
Time:                          21:39:18   Log-Likelihood:                -306.14
No. Observations:                    26   AIC:                             614.3
Df Residuals:                        25   BIC:                             615.5
Df Model:                             0
Covariance Type:              nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       7.754e+04   6286.310     12.335      0.000    6.46e+04    9.05e+04
==============================================================================
Omnibus:                       17.906   Durbin-Watson:                   2.361
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               19.520
Skew:                          -1.932   Prob(JB):                     5.77e-05
Kurtosis:                       4.757   Cond. No.                         1.00
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Figure 12: Sorted Index Nested Loop Join linear model (no intercept)

constant read values and less writes. Since the number of reads dominate the number of writes, this in turn would make the total I/Os constant. A plot of just the writes, as seen in figure 13, suggests that the slope is positive but generally is too noisy to make any statistical inferences. This could be resolved by decreasing the buffer pages, but our DBMS has buffer issues if the number of buffers is less than 100.

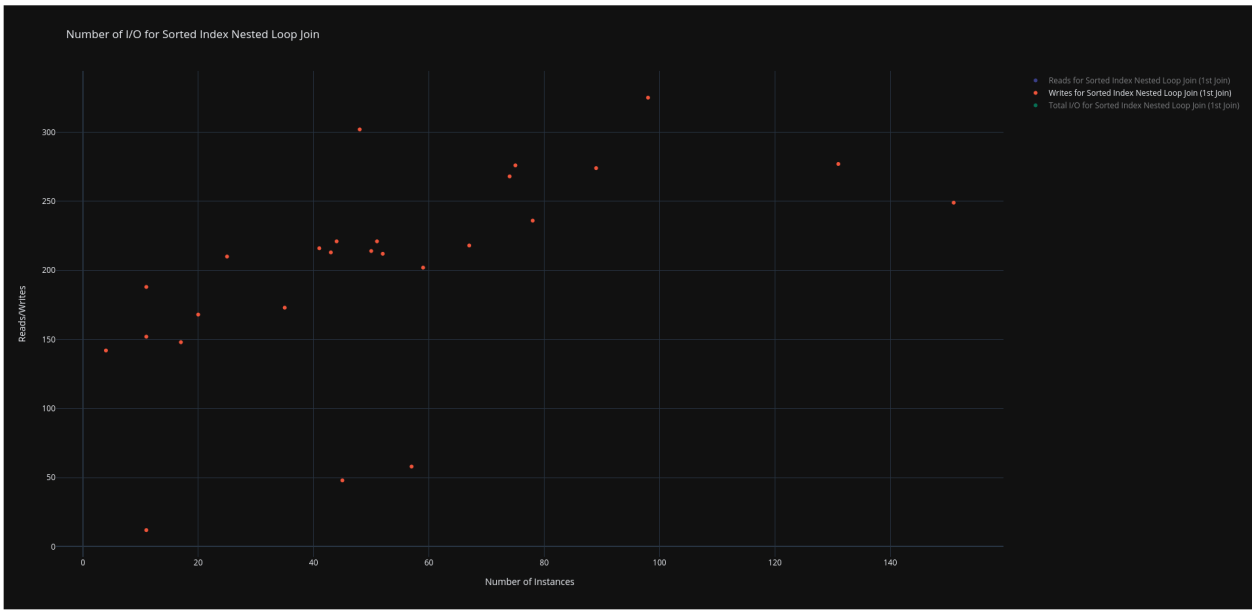

Figure 13: Plot of writes vs. number of filtered instances for sorted index nested loop join

Another explanation is that our search cost is so small that it makes our model appear to be constant. One way to test this would be to actively change which field (i.e., subject, predicate, object, confidence) to sort

by and measure the reads/writes. This should result in different slopes which we can statistically compare. Figure 14 contains the plot of the data with the constant average model.

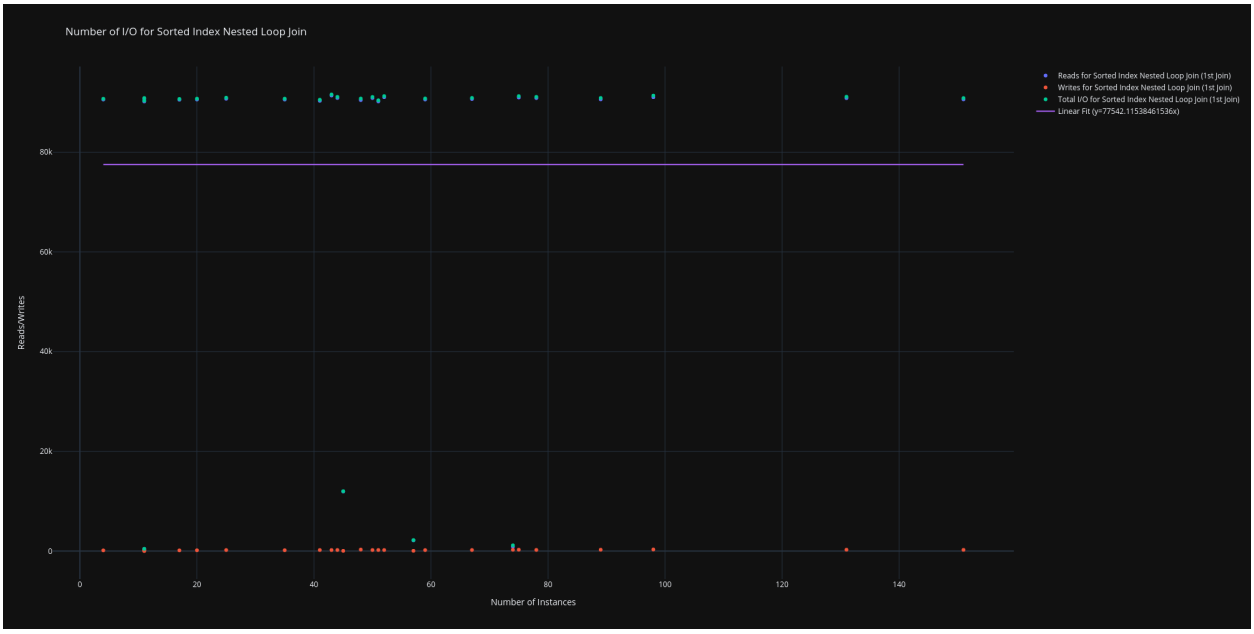

Figure 14: Plot of total reads/writes for index nested loop join

6.4 General Remarks on Experiments

Our experiments seem to support the theoretical time complexities of our join algorithms. More data may be needed to solidify all of these models, particularly for nested loop join and sorted nested loop join. As some of the plots illustrate, there are some outliers that may need to be investigated since the resulting regression model could visually be better. Future work may be to test different inner and outer sizes to see if these algorithms hold up.

References

[1] G.H. Fletcher and P.W. Beck. "Scalable indexing of RDF graphs for efficient join processing." In: *Proceedings of the 18th ACM conference on Information and knowledge management* (Nov. 2009), pp. 1513–1516.

[2] Kun Ren Jiewen Huang Daniel J. Abadi. "Scalable SPARQL querying of large RDF graphs". In: *Proceedings of the VLDB Endowment, Vol 4 Issue 11* (2011), pp. 1123–113.

[3] et al Khadilkar Vaibhav. "Jena-HBase: A Distributed, Scalable and Efficient RDF Triple Store". In: (2012).

[4] Ora Lassila, Ralph R Swick, et al. "Resource description framework (RDF) model and syntax specification". In: (1998).

[5] Abdessamad Belangour Mouad Banane. "An Evaluation and Comparative study of massive RDF Data management approaches based on Big Data Technologies". In: (July 2019), pp. 48–53.

[6] R. Ramakrishnan and J. Gehrke. *Database Management System*. McGraw-Hill, Third edition. McGraw-Hill, 2003. ISBN: 9780198520115.

A Roles of Group Members

1. Pavan Chikkathimmegowda
    a) Command Line Interface Implementation
2. Krima Doshi
    a) BP_Triple_Join: nested loop join, index loop join and sorted index loop join code development
    b) Assisted with triple join integration and testing
3. Tanner Greenhagen
    a) Sorting Functionality
4. Jack Myers
    a) BasicPattern Class
    b) Assisted with triple join integration and testing
5. Shubham Verma
    a) Tuple Modifications: Added features to insert double values and make comparisons for tuples
    b) Assisted with Triple Join approach with Krima
    c) Added Abstract, Introduction, Related Work and References in report
6. Christian Seto
    a) Spearheaded triple join and sort class integration and testing
    b) Ran the experiments for join time complexities
    c) Wrote the conclusion/results