Characterizing Machine Agent Behavior through SPARQL Query Mining

Aravindan Raghuveer
Yahoo!, Bangalore
aravindr@yahoo-inc.com

ABSTRACT

Mining SPARQL queries to understand the behavior of automated programs (or machine agents) is an important step in designing systems for the semantic web. We present techniques that differ from state-of-the-art SPARQL mining techniques in two ways: 1. Move away from *one SPARQL query at a time* view to *SPARQL user session* view 2. Look at the results of SPARQL queries in addition to the query itself. Due to these two approaches, we are able to find two new patterns in SPARQL queries that help us reason better about the underlying program that generated the SPARQL queries. Through a variety of experiments, we show that the patterns found have significant support in all the four datasets provided by the USEWOD committee.

1. INTRODUCTION

The current size of the LOD Cloud, 7.01 billion triples from 219 datasets [1, 2], shows the scale and richness of open linked data available on the web. The LOD Cloud has both human users and automated programs (or machine agents). Recent studies [9, 12, 13, 10, 11, 7] focus on how the datasets are being used by machine agents and human users. This research of understanding the usage patterns is very important especially when designing systems and algorithms that need to scale in multiple aspects: performance, security, stability and availability [12].

SPARQL, a declarative query language, is a popular mechanism to query open linked datasets. In this paper, we present an in-depth study of how machine agents use SPARQL to access open RDF datasets. We focus on detecting patterns in SPARQL queries, a topic of recent interest and research [7]. We present broad access patterns that are common across the four access log datasets made available by the USEWOD committee [8]. Based on the observed access patterns, we reason the characteristics of the underlying program that would have generated the SPARQL queries.

This paper makes three contributions in understanding machine agent behavior:

1. **The SPARQL User Sessions Viewpoint:** Prior work [12, 13, 7] analyze each SPARQL query in isolation to detect patterns. We analyze SPARQL queries in relation to the other SPARQL queries generated by the same user over time. We propose the concept of *SPARQL User Sessions* and provide algorithms to detect user sessions in Section 4. Through the User Session analysis, we show that programs employ two distinct type of loop patterns to scour data from the semantic web. The loop structures are seen to encompass 10% to 95% of the SPARQL queries across the four USEWOD datasets.

2. **Analysis of Results of SPARQL queries:** Unlike prior work that only analyze a SPARQL query, we also look at the results generated by the SPARQL query to characterize the machine agent program that submitted the query. We generate the results by executing the SPARQL query traces on the actual RDF store. Through this extra dimension, we show in Section 5 that machine agent programs aim at finding linkages of the dataset with an authoritative data source like Dbpedia. We see that an average of 7% of the queries per day in the SWDF dataset [3, 8] look for linkage to the dbpedia dataset.

3. **Trends in User Behavior:** Orthogonal to the above contributions, in Section 6 we present interesting query distribution patterns for machine and human users.

The remainder of the paper is organized as follows. We discuss related work in Section 2. In Section 3 we first introduce the USEWOD dataset [8] and also motivate the need for SPARQL query mining. Sections 4 and 5 discuss two key access patterns that we mine from the query logs. We present some interesting aspects of user behavior in Section 6 and Section 7 concludes the paper.

2. RELATED WORK

Query log analysis of servers hosting semantic web content is an important area that is gaining lot of interest in the recent years. Interesting work has been published, thanks to the public datasets like the one made available by the USEWOD committee [8].

Moller et. al [12] provide a strong motivation for understanding SPARQL query patterns. They explicitly call out *semantically aware* agents as a new breed of applications on the semantic web and describe their characteristics. As one of the earliest papers to discuss SPARQL query analysis in

Dataset	#Records (Pre-deduping)	%Records (Post-deduping)	% SPARQL Queries
bio2rdf	192,081	60.75	100
LGD	1,917,052	93.52	100
SWDF	16,693,166	96.81	43.38
Dbpedia	36,204,175	96.74	46.90

Table 1: Summary of the USEWOD datasets. Note that the original datasets of bio2rdf and LGD had only SPARQL queries.

great detail, they also present LOD usage metrics to capture critical aspects of usage pattern analysis. Mario et. al [7] provide further analysis of SPARQL query patterns on the Dbpedia and SWDF datasets [8]. They examine SPARQL queries from both syntactic and structural perspective with specific focus on joins. They show that star shaped joins are the most popular while short chains triple patterns also exist. Francois et al. [13] also provide an in-depth analysis of syntactic structure of SPARQL queries.

On the other hand, Kirchberg et. al [10] analyze log traces from a human user point of view. They introduce the concept of Web Travel Footprints (WTF) which is the trail of the a user's visit to a site. They argue that though a single users WTF does not provide enough information, the sum of WTFs of all users over a period of time provides insight into what aspects of the site are most relevant and how access patterns of the site evolve over time. They introduce a novel visualization method called Kandinsky graphs to show the sum of WTFs on a site.

3. MOTIVATION: SPARQL QUERY MINING

In this section, we first briefly describe the USEWOD 2012 dataset. Next, we motivate through two observations why SPARQL mining is an important step in understanding how machine agents query the semantic web. The first observation is that the volume of SPARQL queries is steadily increasing across two USEWOD datasets (Section 3.2). Second observation points out that a bulk of these queries are most likely generated by programs (Section 3.3).

3.1 The USEWOD 2012 Dataset

The results presented in this paper are based on the USEWOD query access log data [8] for four public datasets: bio2rdf (Linked Data for life sciences), lgd (Linked Geo Data), SWDF (Academic Conference dataset) and dbpedia (Structured knowledge created as part of the Wikipedia project). At a high level, the USEWOD query access log data contains the following information (among other fields that we do not use in our analysis) about each HTTP request received at the servers of each of the 4 datasets named above: timestamp, anonymized IP-Address, Query, User Agent. The Query can be a SPARQL request, RDF request or a plain HTML request. In this paper, we focus on SPARQL requests alone. Please refer [12] for more information about query categorization. The User agent is a string that represents the end-user of the request. We use user agent string heavily in our analysis since it provides very strong hints on whether the end user is a human (browser) or a machine agent (program). For the same end user identified by a combination of the IP Address and User Agent, we remove successive identical queries. Statistics about relative sizes of the four datasets is shown in Table 1.

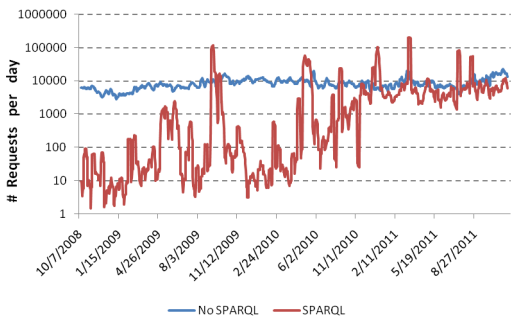

Figure 1: Trend of SPARQL vs Non-SPARQL queries: SWDF

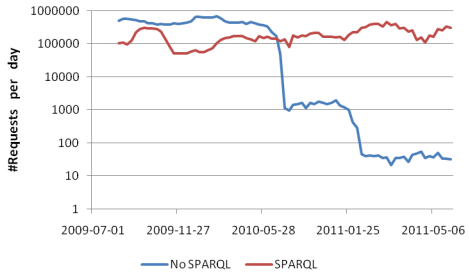

Figure 2: Trend of SPARQL vs Non-SPARQL queries: Dbpedia

3.2 SPARQL Adoption

In the first experiment, we compare the growth rate of SPARQL queries with that of non-SPARQL queries. Figures 1 and 2 show the average number of requests per day for SPARQL and non-SPARQL queries. We chose moving average with a window size of 7 days in order to smoothen the curve. In the SWDF dataset, we can see that the SPARQL query usage has been on the rise since 2008 and towards mid 2011 the number of requests per day of SPARQL and non-SPARQL queries are roughly the same. However, in the dbpedia dataset, the number of non-SPARQL queries declined rapidly after May 2010. We assume that this is an artifact of the log collection process and is not indicative of a real trend. The SPARQL queries in dbpedia have remained consistently high hovering between 0.1 million to a million queries per day. From these two graphs we can conclude that SPARQL based querying of RDF stores is becoming popular and hence an important class of queries to study in depth.

3.3 User Agents for SPARQL queries

In the second experiment, we use Word Clouds [14] as a way to succinctly summarize popular user agents that perform SPARQL queries. Word clouds represent relative popularity of words by using a font size proportional to the frequency of occurrence of the word. Therefore more frequent a word, larger its font size will be. From the dbpedia query logs we scraped the user agent entries of all (roughly 17 million) SPARQL queries . The number of times a user agent appears in the list will be equal to the number of SPARQL queries it sent out. We then used Wordle [4] to create a word cloud as shown in Figure 3. It can be clearly seen that

S.No	Example

1	**Description:** Example of Intra Pattern Loop with varying subject.
	Source: bio2rdf dataset, 6/24/2011
	Two consecutive queries from the loop:
	`SELECT * WHERE {<\protect\vrule width0pt\protect\href{http://bio2rdf.org/dr:D00332}{http://bio2rdf.org/dr:D00332}><http://bi`
	`SELECT * WHERE {<\protect\vrule width0pt\protect\href{http://bio2rdf.org/dr:D00333}{http://bio2rdf.org/dr:D00333}><http://bi`
	Canonical Form of the two queries:
	`SELECT * WHERE { _URI_ _URI_ _URI_ }`

2	**Description:** Example of Intra Pattern Loop with three level nesting. The three varying elements: OPTIONAL block, values of OFFSET, subject of first triple.
	Source: LGD, 5/24/2011
	One query from the loop:
	`SELECT ?b ?v0 FROM <\protect\vrule width0pt\protect\href{http://linkedgeodata.org}{http://linkedgeodata.org}> WHERE { ?b a`
	`?b <\protect\vrule width0pt\protect\href{http://www.w3.org/2003/01/geo/wgs84_pos#lat}{http://www.w3.org/2003/01/geo/wgs84_p`
	`OPTIONAL { ?b <\protect\vrule width0pt\protect\href{http://www.w3.org/2000/01/rdf-schema#label}{http://www.w3.org/2000/01/`
	Canonical Form of the query:
	`SELECT ?v1 ?v2 FROM <\protect\vrule width0pt\protect\href{http://linkedgeodata.org}{http://linkedgeodata.org}> WHERE { _VAR`
	`OPTIONAL { _VAR_ _URI_ _VAR_ . } } OFFSET _LIT_ LIMIT _LIT_`

3	**Description:** Example of Intra Pattern Loop with the CONSTRUCT statement and varying subject.
	Source: bio2rdf, 5/17/2011
	Two consecutive queries from the loop:
	`CONSTRUCT{<\protect\vrule width0pt\protect\href{http://bio2rdf.org/kegg_gene:mca:MCA2270}{http://bio2rdf.org/kegg_gene:mca:`
	`CONSTRUCT{<\protect\vrule width0pt\protect\href{http://bio2rdf.org/kegg_gene:syd:Syncc9605_1989}{http://bio2rdf.org/kegg_ge`
	Canonical Form of the two queries:
	`CONSTRUCT { _URI_ _VAR_ _VAR_ } WHERE { _URI_ _VAR_ _VAR_ }`

4	**Description:** Example of Inter Pattern loop with 2 patterns.
	Source: dbpedia, 09/04/2011
	Two consecutive sets of queries from the loop:
	`SELECT ?x ?p WHERE {?x ?p <\protect\vrule width0pt\protect\href{http://dbpedia.org/resource/Armando_Marsans}{http://dbpedia`
	`SELECT ?p ?y WHERE {<\protect\vrule width0pt\protect\href{http://dbpedia.org/resource/Armando_Marsans}{http://dbpedia.org/r`
	`SELECT ?x ?p WHERE {?x ?p <\protect\vrule width0pt\protect\href{http://dbpedia.org/resource/Eddie_Kasko}{http://dbpedia.org/`
	`SELECT ?p ?y WHERE {<\protect\vrule width0pt\protect\href{http://dbpedia.org/resource/Eddie_Kasko}{http://dbpedia.org/resou`
	Canonical Form of each query pair:
	`SELECT ?v1 ?v2 WHERE {_VAR_ _VAR_ _URI_}`
	`SELECT ?v1 ?v2 WHERE {_URI_ _VAR_ _VAR_}`

5	**Description:** Example of Inter Pattern loop with 2 patterns. An input parameter of the second query is derived from the output of the first query.
	Source: bio2rdf, 6/23/2011
	Two consecutive sets of queries from the loop:
	`SELECT * WHERE { <\protect\vrule width0pt\protect\href{http://bio2rdf.org/cpd:C00022}{http://bio2rdf.org/cpd:C00022}> <h`
	`SELECT * WHERE { ?chebiDrug <\protect\vrule width0pt\protect\href{http://purl.org/dc/elements/1.1/title}{http://purl.org/`
	`<\protect\vrule width0pt\protect\href{http://bio2rdf.org/ns/bio2rdf#image}{h`
	`SELECT * WHERE { <\protect\vrule width0pt\protect\href{http://bio2rdf.org/cpd:C00079}{http://bio2rdf.org/cpd:C00079}> <h`
	`SELECT * WHERE { ?chebiDrug <\protect\vrule width0pt\protect\href{http://purl.org/dc/elements/1.1/title}{http://purl.org/`
	`?chebiDrug <\protect\vrule width0pt\protect\href{http://bio2rdf.org/ns/bio2rdf#image}{http://bio2rdf.org/ns/bio2rdf#image}>`
	Canonical Form of each query pair:
	`SELECT * WHERE { _URI_ _URI_ _VAR_ }`
	`SELECT * WHERE { _VAR_ _URI_ _LIT_ . _VAR_ _URI_ _VAR_ }`

Table 2: Illustrative Examples of Intra Pattern and Inter Pattern Loops. In each row we show example SPARQL queries that are present within loops. Note that the canonical form representation is obtained by replacing actual non-SPARQL-keyword tokens by their types. In the case of Intra Pattern Queries (examples 1 through 3), the canonical form is the same for all the queries within the loop. In the case of Inter Pattern Queries (examples 4,5), multiple canonical query patterns repeat in batches within the loop.

Figure 3: Wordcloud for SPARQL User Agents (Dbpedia). The user agent *Java* appears 4.9 million times

the dominant user agents are programs in Java / Python / Google App engine and so on. We observed a similar pattern in the SWDF logs with the difference that the *ERDF data layer* [5] user agent was also very popular. From the above experiments, we can conclude that SPARQL queries are predominantly used by machine agents that scour the semantic web.

4. LOOP ACCESS PATTERNS

It is well known that loops dominate the instruction trace of modern programs. In the context of SPARQL query logs, we define **loop patterns** as the access log entries that are caused by loops in the program that issued the SPARQL queries. In this section, we show that a significant portion of SPARQL queries across all the four log traces contain loop patterns. First we explain with examples the two type of loops: Intra Pattern (Section 4.2) and Inter Pattern loops (Section 4.3) that are prevalent in access logs. Then in Section 4.4 we present a technique to detect the two type of loops. Section 4.5 presents results of our experiments.

4.1 Definitions

Canonical form of a SPARQL query: Through manual inspection of some USEWOD query logs, we observe in many cases that successive SPARQL queries generated by the same machine agent seem to have an underlying query template. Successive queries generated using the query template have the same structure in the WHERE clause / SELECT statement but values used within the WHERE clause /SELECT statement change. In order to detect the query template, we introduce a canonical form of a SPARQL query whereby we replace every URL, Variable and Literal with the keywords `_URI_, _VAR_,_LIT_` respectively. For example, the canonical form of the query `SELECT * WHERE { ?chebiDrug <http://purl.org/dc/elements/1.1/title> "Pyruvic acid"}` is `SELECT * WHERE { _VAR_ _URI_ _LIT_ }`. Refer Table 2 for sample queries and their canonical forms. We use Apache Jena [6] for all our SPARQL query processing requirements.

User Sessions in SPARQL queries: The requests in the log files provided are sorted by arrival time. We group the requests by the userID and within each user group we sort the requests by arrival time. Based on the task at hand, we define session within the user specific group. Note that userID is a combination of the ipaddress and the user agent string of that user.

4.2 The Intra Pattern Loop

Consider the set of queries shown in the first example of Table 2. The two queries were registered 1 second apart from a user agent *Java/1.6.0_24* on 24/Jun/2011:09:43:53. Due to the request rate and the name of the user agent, we understand that it is a machine agent. All subsequent examples in Table 2 were verified to be generated by programs through the same observations.

It may be observed that the two queries differ only by the subject they are querying for. The subject for the first query being `http://bio2rdf.org/dr:D00332` and `http://bio2rdf.org/dr:D003` for the next. Therefore we see that the canonical forms for both queries look identical as shown in the example. We observe that all the four access logs have such repeating patterns of identical canonical forms for subsequent queries from the same user. We therefore infer that this pattern is generated by a **loop** in the program that iterates over various values of a variable while keeping the query structure constant. We term this access pattern as *Intra Pattern Loop* because a loop occurs within **one** canonical query pattern. The element of the pattern that changes with every query is termed as the *loop variable*. In the above example the loop variable is the subject of the query triple. Likewise, the dataset also contains examples where the loop variable is the predicate or object of the triple.

Example 1 in Table 2 represents the basic structure of an Intra Pattern loop. We observe some very interesting variations that we present in the next sub-sections.

4.2.1 Nested Intra Pattern Loops

The first example had only one loop variable: the subject of the query triple. We observe that Intra Pattern Loops can have multiple loop variables by means of having nested loops. We observe that programs iterate over a range of values for the first loop variable while keeping the values of other loop variables constant. Next a second loop variable's value is changed and the process is repeated. By observing a pattern like this in the access logs, we can reason that the program that generated the logs must have had nested loops to fill values for different loop variables.

An example is shown in Row 2 of Table 2. This example can be found in the LGD dataset with the time stamp 24/May/2011:01:19:23. In this example, we observe that the inner most loop varies the predicate in the *OPTIONAL* part of the query. The *only* three values used by the predicate variable are: `http://www.w3.org/2000/01/rdf-schema#label`, `http://www.w3.org/2003/01/geo/wgs84_pos#long`, `http://www.w3.org/`

The second loop varies the value of *OFFSET* by increasing it by 1000 every time. The final outer most loop varies the object of the first triple. Some examples of the values taken by the variable are `http://linkedgeodata.org/ontology/PlaceOfWorshi` `http://linkedgeodata.org/ontology/School`.

4.2.2 Type of the Loop Variable

We observe that values assumed by loop variables can be both Quantitative and Categorical. Quantitative variables assume numerical values like the *LIMIT* in the second example in Table 2. Categorical variables assume values from a finite set of values. In the first example in Table 2 the loop variable is seen to be categorical. The finite set of values could either be supplied through a dictionary or are a result of a previous query by the same user.

4.2.3 Query Type in Intra Pattern Loops

We observe that Intra Pattern Loops are present across different query types. Example 3 in Table 2 shows an Intra

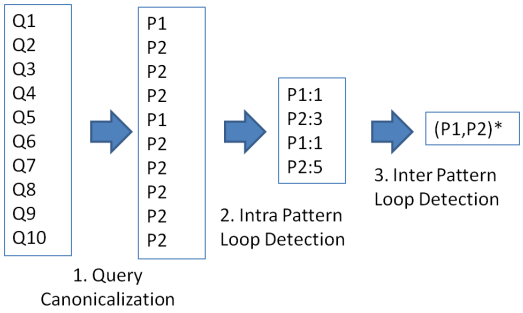

Figure 4: Steps in Loop Detection within a user session

Pattern loop with a *CONSTRUCT* query. We have seen cases of *ASK* queries in Intra Pattern loops as well.

4.3 The Inter Pattern Loop

The next family of loop patterns that we observe in all the four access logs is called the *Inter Pattern Loop*. In the Inter Pattern loop, the program iterates over a set of canonical patterns repeatedly. Row 4 in Table 2 shows an example Inter Pattern Loop. In this example, from the dbpedia dataset with the timestamp 09/Apr/2011 04:00:00, we see that the program first gets all relationships for an an entity (for instance, Armando Marsans) in the object role. Next, it gets all the relationships for the same entity, this time the entity playing the subject role. Then this pair of query is iterated over a large set of entities.

Another interesting variation of the Inter Pattern Loop is shown in row 5 of Table 2. This example was observed in the bio2rdf dataset with the time stamp 24/Jun/2011:09:47:37. In Example 5, the input to the second query in the pattern is supplied by the result of the first query. We see that the first query gets all the information for the compound denoted by the URI `http://bio2rdf.org/cpd:C00022`. Then it uses the compound name information from the previous result (*Pyruvic Acid* in this case) as a parameter to the next query to retrieve the drug information and the compound structure. We verified this pattern by sifting through multiple instances of this occurrence. We find this pattern of querying very significant because it shows programs are becoming sophisticated enough to pipeline queries.

We close the description of Intra and Inter Pattern Loops with one remark. The Intra and Inter Loop examples we described in Sections 4.2 and 4.3 are **not** exhaustive in that they cover all possible manifestations of inter and intra loops. The examples are primarily shown for illustrative purposes.

4.4 Loop Detection Technique

In this section, we describe an automated method to detect Intra Pattern and Inter Pattern Loops in all the logs. The proposed method is independent of the dataset. Figure 4 shows the three steps involved in detecting Intra and Inter Pattern loops. the input to the first step is a list of queries for a user ordered by arrival time of the query. First, we canonicalize the query to a standard form. In the second step, we form groups of canonical patterns. For two consecutive queries to be part of a canonical group, two conditions must be met: 1. They have the same canonical pattern. 2. They have at least one URI or LITERAL whose value remains the same across the two queries. This is to ensure

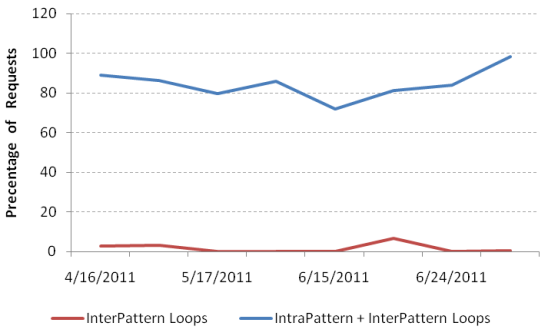

Figure 5: Loop Coverage: Bio2RDF

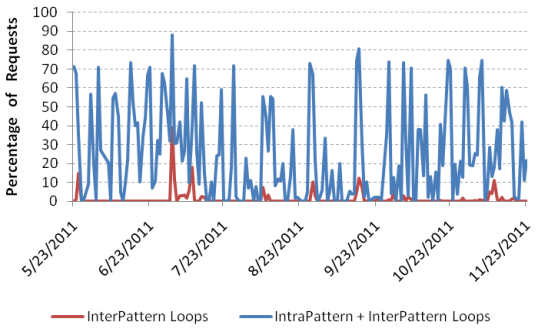

Figure 6: Loop Coverage: LGD

that we do not group dissimilar queries together just because they had the same underlying canonical pattern. We also store the support for each pattern.

The input to the third step now does not have repetitions of patterns as shown in Figure 4. The Inter Pattern loop detection does not use the support information. It takes as input a sequence of patterns and finds repeating blocks within the sequence. We use a simple technique to detect possible loops. The proposed algorithm does not work for all cases of looping but is sufficient for our needs to detect simple and nested loops. The loop detection proceeds as follows: The sequence is scanned from left to right. If a pattern is seen for the second time, tokens appearing between the two occurrences of the pattern could be a potential loop. This hypothesis is tested and the loop boundaries noted. We leave to future work a more sophisticated loop detection algorithm to mine more loop patterns.

4.5 Experiments

In this section, we present the results of our experiments related to Intra and Inter Pattern Loop mining on all the four datasets. We used map-reduce on the Hadoop platform to perform all the experiments at scale.

4.5.1 *Coverage of Intra and Inter Pattern Loops*

In the first experiment, we aim to find the prevalence of Intra Pattern and Inter Pattern loops in the four datasets. Figures 5 through 8 shows for the four datasets, the percentage of SPARQL queries that are part of either an intra pattern or inter pattern loop. Each point in the graph summarizes the contribution of one day's worth of access logs.

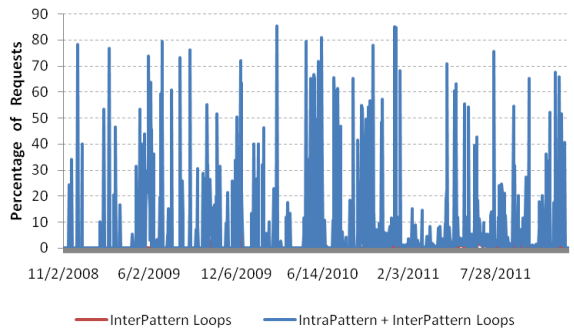

Figure 7: Loop Coverage: SWDF

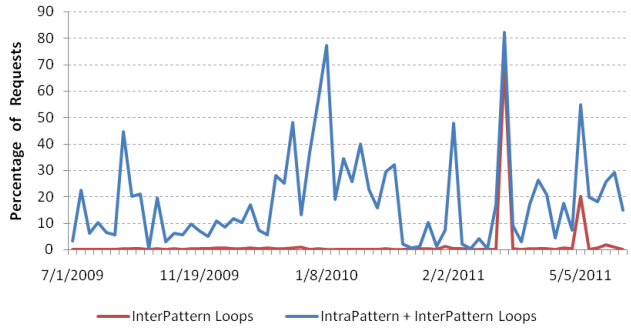

Figure 8: Loop Coverage: DBpedia

Across all the four datasets, we can make two general observations: 1) Intra Pattern and Inter Pattern loops form a significant chunk of access patterns of machine driven SPARQL queries 2) Intra Pattern Loops are far more common than Inter Pattern loops.

Figure 5 shows that the prevalence of loops in the bio2rdf access logs is very high, touching almost 100% on 2011-06-25. In the LGD and SWDF datasets we observe that the inter pattern loop coverage is very low (around 1% per day). Dbpedia on the other hand (refer Figure 8) has slightly better inter loop coverage (2.5% per day). As seen in Figure 8, on 2011-04-09 we observe a large spike in inter pattern coverage. The underlying reason for this spike is the set of queries generated by one user. The inter pattern loop consisted of two query patterns shown in Example 4 in Table 2. These two patterns covered 725,350 SPARQL queries out of 1,570,833 SPARQL queries registered on 2011-04-09. This contributes to 46% of the total queries on 2011-04-09.

4.5.2 *Intra Pattern Loop Characteristics*

In the next set of experiments, we summarize the characteristics of intra pattern loops across the four datasets through the following four metrics:

1. **Average Number of Intra Pattern Loops per day:** While the previous experiment measured the coverage of intra pattern loops, this metric aims to quantify the frequency of occurrence of intra pattern loops. For example, in the sequence of patterns shown in Figure 4, four instances of intra pattern loops can be observed as output of Step 2.

2. **Average Number of Loop Variables per loop per day:** The number of loop variables in an intra pattern signify the level of nesting that happens in an intra loop. In example 2 in Table 2, the number of loop variables is three. First, the OPTIONAL block, second the LIMIT variable and finally the object of the first triple in the WHERE clause.

3. **Average Cardinality per loop variable per day:** The cardinality of a variable is defined as the number of values that can be taken by a loop variable.

4. **Average Number of Unique Patterns per day:** This metric presents another view of the coverage of loops. If the total number of unique patterns per day is very low it signifies that many patterns repeat frequently and thereby implying that loops are common in the dataset.

Figure 9 shows the comparison of the four datasets with respect to the four metrics defined above. Note that y-axis is in log-scale. SWDF seems to have the least amount of inter pattern loops per day while dbpedia has the maximum. This is also a direct consequence of number of SPARQL requests seen in the datasets. The average number of variables per loop is very similar across the four datasets (mean = 1.36 and standard deviation = 0.27). Therefore we see that the nesting in intra pattern loops is not very common. The average cardinality per variable is extremely high for the bio2rdf dataset in comparison to the other 3 datasets. The cardinality per variable of the bio2rdf dataset is 1565 per variable as compared to an average of 63.7 values/variable (standard deviation=37.01) for the other 3 datasets. We correlate this observation to fact that the bio2rdf dataset exhibits systematic querying of a large number of the drugs in the Chebi drug bank.

4.5.3 *Inter Pattern Loop Characteristics*

Similar to the previous section, we now define three metrics to characterize the behavior of Inter Patter Loops in the four datasets.

1. **Average Number of Inter Loops per day:** This metric aims to capture the frequency of inter loops in each dataset.

2. **Average Loop Depth per loop per day:** This metric is similar to the *Average Number of Loop Variables per loop* in the Intra Pattern metrics. It summarizes the level of nesting in inter pattern loops.

3. **Fraction of Inter Pattern Loops that had Intra Pattern Loops:** This metric is designed to answer the question: Do inter pattern loops always embed within them one or more intra pattern loops?

Figure 10 shows that the bio2rdf dataset has the highest number of inter pattern loops per day. We attribute this to one specific kind of pattern shown in Row 5 of Table 2. We also need to cross-correlate this observation with the another observation in Figure 5 that the number queries covered by inter pattern loops is still very low. The reason for this disparity is because the the number of patterns in the loop is 2 and hence even a large number of loops does not contribute significantly to the total number of queries covered by inter pattern loops.

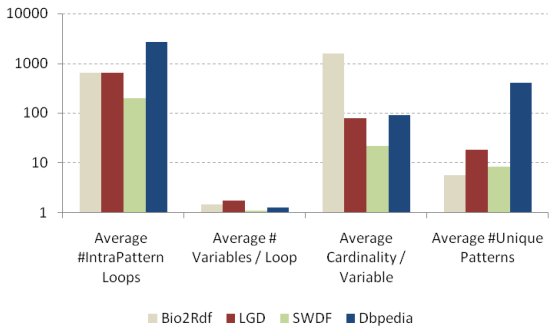

Figure 9: Intra Pattern Loop Parameters

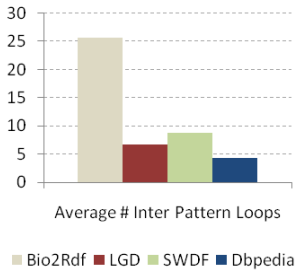

Figure 10: Inter Pattern Loop Parameters

Figure 11 shows that deep nested loops are not very prevalent in all 4 datasets. It also makes an interesting observation that roughly 50% of the times, an inter pattern loop always embeds an intra pattern loop.

5. QUERYING LINKAGE WITH DBPEDIA

Research in the past has found that DBpedia is one of the key hubs in the open data graph [12, 2]. A majority of the data sources link directly into dbpedia. We hypothesize that since DBpedia is an authoritative source of knowledge, programs will want to discover links to dbpedia wherever possible. In the next experiment, we investigate if this hypothesis is reflected in the query logs. We use the SWDF dataset for this experiment. This requires two conditions to be met: **(a)**The dataset (SWDF) should have links to Dbpedia **(b)** The programs should be querying to get the attributes that have dbpedia links.

We first downloaded the RDFs of the SWDF dataset [3].

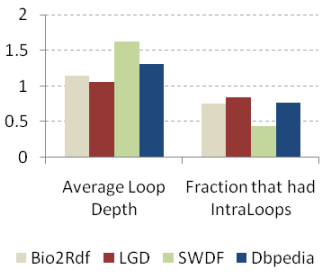

Figure 11: Inter Pattern Loop Parameters

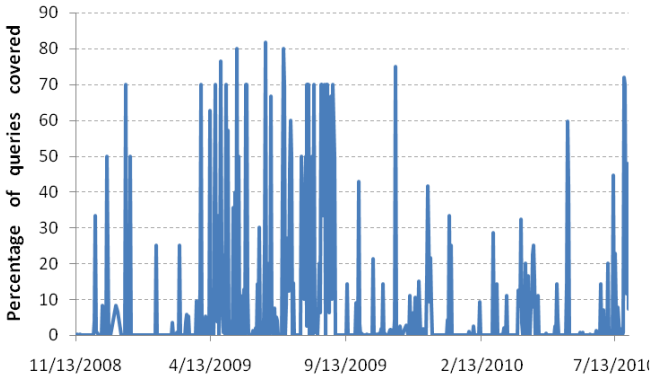

Figure 12: Percentage of queries per day that have at least one Dbpedia result

We verified through simple word search program that condition **(a)** above is met. Next we replayed 20 months worth of queries (November 2008 to July 2010) on the local RDF files. We force a limit of 15,000 to the number of results for every query and we execute only SELECT queries. For each query, we examined the results to check if at least one dbpedia URL was returned. Figure 12 shows that a significant percentage of queries per day had at least one dbpedia URL in the result. The mean was found to be 7.94% and standard deviation 17.96. There are 6 instances where the percentage is over 70%.

In order to ensure that the above finding is not specific to the SWDF dataset, we repeated the experiment on the LGD dataset. However, instead of executing the SPARQL queries on a local RDF database, we executed them on a remote SPARQL endpoint(`http://linkedgeodata.org/sparql`). In order to respect politeness and more importantly not to introduce spurious access log entries at the SPARQL end point, we replayed the `SELECT` SPARQL queries from the log traces for 2 days worth of data (2011-07-14, 2011-07-15).

We found that out of 414 queries that we issued, roughly 26.5% of them had atleast one dbpedia URI in their result. Further, we investigated the underlying reason behind finding the dbpedia linkages. Consider this example query from the 2011-07-14 log issued by a user agent Java/1.6.0_23:

```
SELECT * WHERE {?country <\protect\vrule width0pt\protect\hr
/02/22-rdf-syntax-ns#type> <\protect\vrule width0pt\protect\hre
ontology/Country> .?country <\protect\vrule width0pt\protect\hr
property/is_in> "Europe" .?country <\protect\vrule width0pt\pro
2002/07/owl#sameAs> ?countryInDbpedia}
```

The query seems to be getting the list of all European countries and their corresponding link from dbpedia. We think the reason behind this behavior is that a Dbpedia link is an authoritative source that the program wants to link its results to. Also since Dbpedia being a hub in the LOD cloud [12], more information from other sources can be found from other sources that link to the same Dbpedia entry.

Another important observation we make from the above example is that the query by itself does not have enough information to be able to discover this pattern. The proposed method of executing the query and analyzing the results led us to finding this interesting linkage pattern.

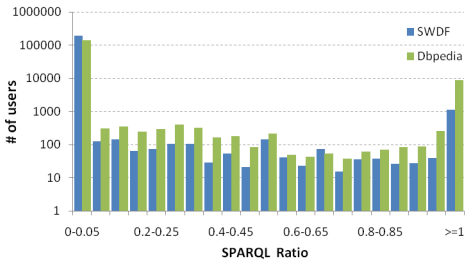

Figure 13: SPARQL Ratio Histogram

6. PATTERNS IN USER BEHAVIOR

In Section 3.3 we briefly studied the behavior of users with respect to SPARQL queries. In this section, we present a broader perspective of user behavior.

For the experiments described in this section, we use all the datasets except the DBPedia 3.5.1 and the bio2rdf datasets. The DBpedia 3.5.1 and the bio2rdf datasets have anonymized ipaddress and user agent values which hinder any user specific analysis. We assume a user is uniquely identified by the combination of the IP Address and the user agent.

In the first experiment, we aim to find an answer to the following question: "Do users (machine or human) use a combination of SPARQL and non-SPARQL queries?" To this end, we define a metric called SPARQL ratio S_i for a user i.

$$S_i = \frac{N_i^{sparql}}{(N_i^{sparql} + N_i^{non-sparql})} \quad (1)$$

where N_i^{sparql} is the number of SPARQL queries made by user i and $N_i^{non-sparql}$ is the number of non-SPARQL queries made by the same user. We calculated the SPARQL ratio for all users in the dbpedia and the swdf dataset. Figure 13 shows this histogram with the x-axis being 20 SPARQL ratio intervals and y-axis being the number of users in each interval. Note that the y-axis is in log-scale. We see that first ($0 \leq S_i < 0.05$ and last interval ($S_i \geq 1$) have the maximum support. All the intermediate interval have very low support. This shows that a majority of users either use SPARQL queries or non-SPARQL queries and not a combination of them.

Next, we make a very interesting observation across the three datasets: *A very small set of users contribute to a large percentage of the queries* (Refer Table 3). Based on the above observation, we define two buckets of users. The first bucket of users are the top contributors that account for 90% of the queries. The next bucket of users account for the remaining bottom 10% of the queries. We hypothesize that the top 90% bucket comprises mostly of machine agents while the bottom 10% bucket comprises mostly of human users.

In the next experiment, we calculate the average SPARQL Ratio for all users belonging to each bucket. In Table 3 we see that the average SPARQL ratio that we defined in Equation 1 is an order of magnitude higher for the Top 90% bucket. This can be explained by correlating with an earlier observation that SPARQL queries are dominated by programs (from Figure 3). Also, SPARQL ratio of the LGD dataset is 1 because this dataset comprises of only SPARQL queries. This additional observation further re-inforces our theory that top 90% bucket is dominated by machine agents.

Dataset	% Requests	#of users	SPARQL Ratio
LGD	Top 90%	18	1
	Bottom 10%	1084	1
DBPedia	Top 90%	644	0.46
	Bottom 10%	154096	0.06
SWDF	Top 90%	1678	0.05
	Bottom 10%	191335	0.007

Table 3: Request Distribution: 90% of the requests are contributed by 0.4% of the total users in the dbpedia dataset, 1.6% of users in the LGD dataset and 0.8% of the users in the SWDF dataset.

7. CONCLUSION

In this paper we presented two new ways of mining the SPARQL query log data. Through SPARQL query mining we showed that machine agents often use loops to query the datasets on the LOD cloud. We presented seven new metrics to model loop usage. We also showed by analyzing query results from two datasets (SWDF and LGD) that machine agents often try to find linkages to Dbpedia data. These two findings have hence improved our knowledge on how machine agents operate on the LOD cloud.

8. REFERENCES

[1] `http://stats.lod2.eu/`.
[2] `http://richard.cyganiak.de/2007/10/lod/`.
[3] `http://data.semanticweb.org/dumps/`.
[4] `http://www.wordle.net/`.
[5] `http://www.erdf.nl/`.
[6] `http://incubator.apache.org/jena/`.
[7] M. Arias, J. D. Fernández, M. A. Martínez-Prieto, and P. de la Fuente. An empirical study of real-world sparql queries. *CoRR*, abs/1103.5043, 2011.
[8] B. Berendt, L. Hollink, V. Hollink, M. Luczak-Rösch, K. H. Möller, and D. Vallet. Usewod2012 Ů 2nd international workshop on usage analysis and the web of data. In *In 21st International World Wide Web Conference (WWW2012), Lyon, France.*
[9] M. Hausenblas. Exploiting Linked Data For Building Web Applications. *IEEE Internet Computing*, N(N):N–N, 2009.
[10] M. Kirchberg, R. K. L. Ko, and B.-S. Lee. From linked data to relevant data – time is the essence. *CoRR*, abs/1103.5046, 2011.
[11] M. Luczak-Rosch and M. Bischof. Statistical analysis of web of data usage. In *Joint Workshop on Knowledge Evolution and Ontology Dynamics (EvoDyn2011).*
[12] K. Möller, M. Hausenblas, R. Cyganiak, and G. A. Grimnes. Learning from linked open data usage: Patterns and metrics. In *Proceedings of the WebSci10: Extending the Frontiers of Society On-Line*, 2010.
[13] F. Picalausa and S. Vansummeren. What are real sparql queries like? In *Proceedings of the International Workshop on Semantic Web Information Management*, SWIM '11, pages 7:1–7:6, New York, NY, USA, 2011. ACM.
[14] A. W. Rivadeneira, D. M. Gruen, M. J. Muller, and D. R. Millen. Getting our head in the clouds: toward evaluation studies of tagclouds. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '07, pages 995–998, New York, NY, USA, 2007. ACM.