Traffic Analytics for Linked Data Publishers

Luca Costabello[1(✉)], Pierre-Yves Vandenbussche[1], Gofran Shukair[1], Corine Deliot[2], and Neil Wilson[2]

[1] Fujitsu Ireland Ltd., Galway, Ireland
{luca.costabello,pierre-yves.vandenbussche, gofran.shukair}@ie.fujitsu.com
[2] British Library, London, UK
{corine.deliot,neil.wilson}@bl.uk

Abstract. We present a traffic analytics platform for servers that publish Linked Data. To the best of our knowledge, this is the first system that mines access logs of registered Linked Data servers to extract traffic insights on daily basis and without human intervention. The framework extracts Linked Data-specific traffic metrics from log records of HTTP lookups and SPARQL queries, and provides insights not available in traditional web analytics tools. Among all, we detect visitor sessions with a variant of hierarchical agglomerative clustering. We also identify workload peaks of SPARQL endpoints by detecting heavy and light SPARQL queries with supervised learning. The platform has been tested on 13 months of access logs of the British National Bibliography RDF dataset.

Keywords: Linked data · Traffic analytics · Data publication · SPARQL

1 Introduction

Data providers have so far published hundreds of linked datasets, thus contributing to the birth of the Web of Data. Although technical and conceptual challenges of Linked Data publication have been largely discussed in literature [2,10], we believe that data publishers have still limited awareness of how datasets are accessed by visitors. In other words, data providers cannot easily find valuable insights into how triples are consumed and by whom. This has two consequences: first, publishers struggle to justify Linked Data investment with management. Second, they miss out technical benefits: poor access traffic knowledge hampers negotiating an affordable service level agreement with hosting suppliers; limited awareness of traffic spikes prevents predicting peaks during real-world events. While some works describe specific access metrics for linked datasets [13], no comprehensive analytics tool for Linked Data publishers has ever been proposed, and in most cases publishers have no choice but to manually browse through records stored in server access logs. Applications for analysing traditional websites traffic exist, but none considers the specificities of Linked Data: none extracts insights by parsing SPARQL in query strings (e.g. how many resources occur in SPARQL queries, how many SPARQL endpoints workload peaks, the ratio of HTTP and SPARQL traffic, etc.). Besides, content negotiation with `303` URIs[1] is not interpreted as an atomic operation, thus overestimating the actual number of requests. No tools detect Linked Data visitors sessions, neither.

Bringing traffic analytics to Linked Data publishers requires solving three major challenges: first, we must choose meaningful metrics: are traditional web traffic analytics metrics sufficient? How should we tailor such metrics to Linked Data? Second, we must extract such metrics. This leads to additional sub-problems: which data sources shall we mine, and how do we filter noise (e.g. search engine crawlers, robots)? Some metrics require knowledge of visitor *sessions*: how to detect such sessions in a Linked Data scenario? Being aware of workload peaks of SPARQL endpoints is important: how do we detect *heavy* SPARQL queries repeatedly issued in a short time? Last but not least, how do we deliver insights to end users without human intervention?

Research Contribution. Our contribution is twofold: first, we present a traffic analytics platform for Linked Data servers: we add novel Linked Data-specific metrics - to break down traffic by RDF content, capture SPARQL insights, and properly interpret 303 patterns. Second, we describe two mining tasks that the system adopts to extract some of the aforementioned metrics: (i) the reconstruction of Linked Data visitors sessions with time-based hierarchical agglomerative clustering, and (ii) the detection of workload peaks of SPARQL endpoints by predicting heavy and light SPARQL queries with supervised learning and SPARQL syntactic features. We evaluate our mining tasks on DBpedia 3.9 and British National Bibliography logs, and we publish the datasets to reproduce our results. We describe the insights from 13 months of access logs of the British National Bibliography dataset[2], and we show that our system reveals patterns that would otherwise require heavy manual intervention by dataset publishers.

The paper is organised as follows: Sect. 2 lists related works. Section 3 introduces the system architecture. Section 4 discusses the traffic metrics and their extraction. This section also describes how we identify visitor sessions, and how we detect heavy and light SPARQL queries. Results and reproducibility are described in Sect. 5. Section 6 proposes future extensions.

2 Related Work

Google Analytics[3] and other popular web analytics platforms[4] (e.g. Open Web Analytics, PIWIK[5]) are not designed for linked datasets (e.g. they do not provide

[1] https://www.w3.org/TR/cooluris.
[2] http://bnb.data.bl.uk.
[3] http://analytics.google.com.
[4] https://en.wikipedia.org/wiki/List_of_web_analytics_software.
[5] http://piwik.org, http://www.openwebanalytics.com.

access metrics for SPARQL). Many works over the last decade discuss access metrics for the traditional web [6]. Nevertheless, only a handful propose Linked Data-specific traffic metrics: Möller et al. [13] propose a list of Linked Data-specific metrics that cover HTTP and SPARQL access to RDF (e.g. ratio between `303` and 200 HTTP requests, number of *RDF-aware* agents, SPARQL query features, machine/vs human classification based on user-agent strings). The well-established USEWOD workshop series[6] is the reference for Linked Data usage mining (the workshop authors also publish a dataset of anonymised linked datasets access logs [11]). We reused and extended metrics defined in [6,13].

Labeling SPARQL queries as light or heavy has been inspired by [5,7,9,18]. Although formal analysis of the SPARQL language complexity has been extensively studied [16,19], empirical works often adopt an informal notion of query complexity: for instance, Möller et al. [13] consider complexity as the ratio between the triple pattern types and the count of SPARQL queries. Others [4] refer to the count of classes and properties of a query. Other empirical studies propose syntactic features that distinguish low from high complexity queries, and confirm that non-conjunctive queries lead to higher execution times [5,7,18]. We trained our light/heavy SPARQL classifier with syntactic features proposed in these papers. Other works use supervised learning on SPARQL query logs: Hasan [9] goes as far as predicting SPARQL query execution time with k-nearest neighbours regression and support vector machines.

To the best of our knowledge, no existing work detects Linked Data client sessions. Traditional web sessions have been naively determined by identifying fixed-length inactivity gaps, but picking optimal values is awkward and error-prone [1]. Murray et al. present a variant of hierarchical agglomerative clustering (HAC) - later refined by [12] - that finds visitor-specific thresholds [14]. We adopt HAC in our Linked Data scenario. Other clustering approaches rely on model-based cluster analysis [15] and time-aware clustering [8,17].

3 System Overview

The traffic analytics platform includes the following main components (Fig. 1):

Extract-Transform-Load (ETL) Unit. On a daily basis, for registered publishers, the Log Ingestion sub-component fetches and parses access logs from one or more linked dataset servers (see Fig. 2 for an example). Since data publishers adopt different access records formats, the platform relies on a flexible and customisable log parser. Records are filtered from search engines crawlers noise.

Metrics Extraction Unit. Extracts traffic metrics. It includes visitor session detection, and light/heavy SPARQL query classification (Sect. 4). Note that we do not support structured content embedded in HTML pages (e.g. microdata, RDFa). The system does not provide statistics on downloads of datasets dumps.

[6] http://usewod.org/.

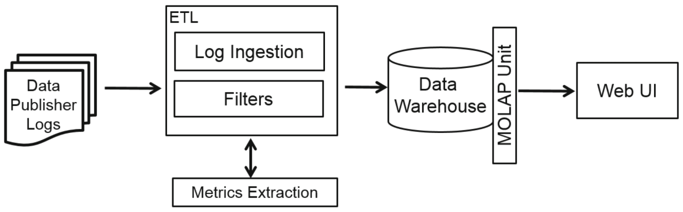
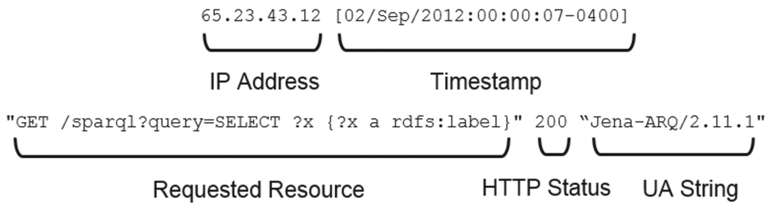


Fig. 1. Architecture of the traffic analytics platform for linked data publishers



```
65.23.43.12 [02/Sep/2012:00:00:07-0400]
"GET /sparql?query=SELECT ?x {?x a rdfs:label}" 200 "Jena-ARQ/2.11.1"
```

Fig. 2. A record of a linked data server access log (Apache commons logfile (https://httpd.apache.org/docs/trunk/logs.html#common)).

Data Warehouse and MOLAP Unit. Traffic metrics are stored in a data warehouse equipped with an SQL-compliant MOLAP[7] unit that answers queries with sub-second latency.

Web User Interface. The front end queries the RESTful APIs exposed by the MOLAP Unit, and generates a web UI that shows the metrics in Table 1 filtered by date, user agent type, and access protocol (see screenshots in Fig. 3).

4 Traffic Metrics

The traffic analytics platform features three categories of traffic metrics:

Content Metrics. How many times RDF resources have been accessed. We interpret content negotiation with 303 URIs as an atomic operation. We also count how many times resource URIs appear in SPARQL queries[8], and provide aggregates by *family* of RDF resource (i.e. instances, classes, properties, graphs).

Protocol Metrics. Information about the data access protocols used by visitors. Includes SPARQL-specific metrics such as the count of malformed queries and the detection of light and heavy SPARQL queries with supervised learning.

[7] Multidimensional Online Analytical Processing.
[8] This is a lower bound estimate. Access logs do not contain SPARQL result sets.

Table 1. Supported metrics.

Content Metrics	
Instances[a]	How many times RDF instances have been requested (HTTP requests and URIs in SPARQL queries)
Classes[a]	The count of URIs used as RDFS/OWL classes in SPARQL queries. URIs must be objects of `rdf:type`.
Properties[a]	The count of URIs used as predicates in SPARQL queries.
Graphs[a]	The count of URIs used as graphs in SPARQL queries (`FROM/FROM NAMED`, `USING/USING NAMED`, `GRAPH`).
Protocol Metrics	
Data access protocol[a]	The count of HTTP lookups and SPARQL queries, over a specific time frame.
SPARQL query type[a]	The count of SPARQL verbs over a specific time frame (e.g. how many `SELECT`, `ASK`, `DESCRIBE`, and `CONSTRUCT`).
Light/Heavy SPARQL[a]	Indicates the number of *light* and *heavy* SPARQL queries (see Sect. 4.2 for details).
HTTP methods	Indicates the count of HTTP verbs (e.g. `GET`, `POST`, `HEAD`).
303 Patterns[a]	The count of HTTP `303` patterns found in logs (`303` URIs).
Misses	Keeping track of HTTP `404`s is useful to understand whether visitors are looking for resources which are not currently included in the dataset.
Malformed queries[a]	The count of `HTTP 400`s shows how many malformed HTTP requests have been issued. We also show how many SPARQL queries contain syntax errors.
Other client-side errors	Useful, for example, to detect whether visitors attempt to access forbidden RDF resources (`HTTP 403`).
Server-side errors	The count of `HTTP 5xxs` is important to estimate whether error-triggering SPARQL queries have been repeatedly issued to a triplestore.
Audience Metrics	
Location	Continent, country, subdivision, and city of origin of a visitor.
Network provider	The name of the company or institute associated to a visitor's IP address, as returned by WHOIS lookups. If WHOIS data is not available, we provide the visitor's host network.
Language[a]	The preferred language requested by a visitor. Such information is extracted from the `Accept-Language` HTTP header (for HTTP lookups) and retrieved from SPARQL `xsd` language-tagged string literals (e.g. `@en`) or `FILTER lang()`s (e.g. `lang(?l) = "en"`).
User agent	The user agent string provided by a visitor.
User agent type	To provide a clearer estimate of which clients are used to access a dataset, we group user agent strings into Software Libraries (e.g. Jena, Python SPARQL-client, etc.), Desktop Browsers, Mobile Browsers, and Others.
New vs Returning	New visitors versus visitors that have performed at least one visit before.
External referrer	HTTP `Referer:` headers. When dereferencing an RDF resource, the HTTP request might contain this optional header field that specifies the URI from which the request has been issued.
Sessions count	The count of all visitors sessions.
Session size	The number of requests sent by a visitor during a session (requests might be a mix of HTTP lookups and SPARQL queries).
Session depth[a]	The number of distinct RDF resources (graphs, classes, properties, instances) requested by a visitor during a session.
Session duration	The duration of a session.
Bounce rate	Indicates the percentage of sessions that contain only one resource request (whether this is an HTTP lookup or a SPARQL query).

[a] indicates linked data-specific entries

Audience Metrics. Besides traditional information about visitors (e.g. location, network provider), these measures include details of visitor sessions.

Table 1 describes the three categories, and highlights Linked Data-specific metrics (e.g. *light*/*heavy* SPARQL queries). In the following sections we describe how metric extraction is carried out by the system.

4.1 Content Metrics Extraction

Our traffic analytics platform supports Linked Data dual access protocol: it counts how many times an RDF resource is dereferenced with HTTP operations, but also how many times its URI is included in SPARQL queries. Unlike existing tools, we interpret content negotiation with 303 URIs as an atomic operation, thus counting each HTTP 303 pattern as a single request. Table 1 shows the list of aggregates by *family* (instances, classes, properties, graphs). While extraction of resources and their family is straightforward for HTTP operations (target URIs belong by default to the *instance* family), for SPARQL it requires parsing each query to determine the family of a resource (see Table 1 for parsing details).

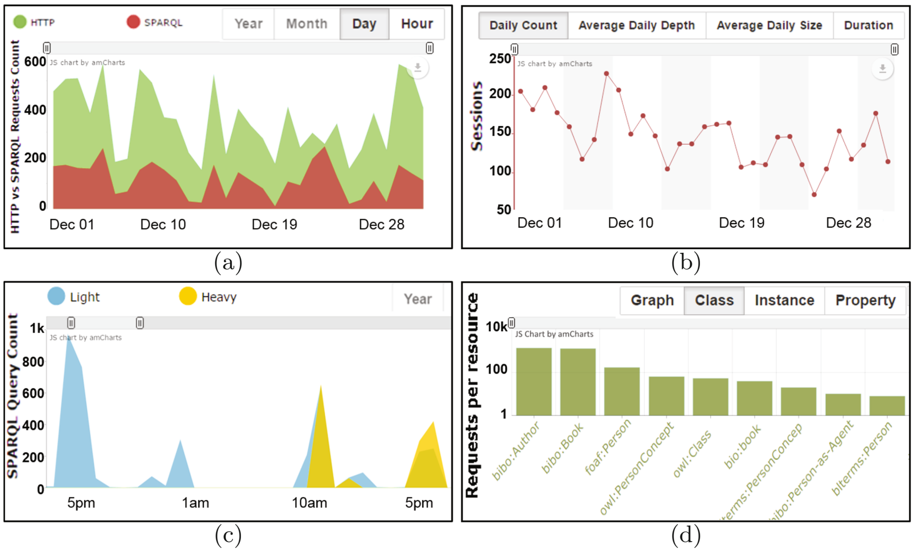


Fig. 3. Screenshots of the web UI: (a) HTTP vs SPARQL; (b) sessions insights; (c) light and heavy SPARQL queries spikes; (d) most popular RDF classes.

4.2 Protocol Metrics Extraction and SPARQL Queries Weight

Besides the count of HTTP lookups, 303 patterns, malformed queries, etc., this group includes the count of *light* and *heavy* SPARQL queries. We now describe the problem determined by this task, followed by our contribution.

```
SELECT ?x
WHERE
{?x a foaf:Person.}
```

(a)

```
SELECT ?u ?c ?l
WHERE {{
  ?u rdfs:comment ?c .
  FILTER(regex(str(?c), '(^|\\\\W)fr', 'i'))}
 UNION {
  ?u rdfs:label ?l .
  OPTIONAL {?u skos:prefLabel ?l.}}}
```

(b)

Fig. 4. Sample SPARQL queries. *light* (a), and *heavy* (b). Note the presence of `UNION`, `OPTIONAL`, and `FILTER regex` in query (b).

Detecting Heavy and Light SPARQL Queries. We argue that identifying workload peaks of a SPARQL endpoint requires a rough estimate of how many *heavy* SPARQL queries have been sent to a SPARQL endpoint. A number of challenges arise: first, an (informal) definition of *weight* of a SPARQL query is required (i.e. when is a query *heavy* or *light*?). Second, a method to detect heavy and light queries is needed. Identifying *heavy* and *light* queries can be cast as a supervised binary classification problem; however, supervised learning requires a training and test dataset, and access logs collections usually do not include any information on execution time, least of all a manual annotation of *light*/*heavy* queries. Third, a comprehensive, established, and unified list of discriminative features for this specific classification task has not been defined yet, although features have been proposed for similar tasks (see related work in Sect. 2).

We use SPARQL syntactic features to train off-the-shelf supervised binary classifiers that label SPARQL queries in two categories: *light* and *heavy* (see Fig. 4 for an example). Although such categories can be more thoroughly explained by means of SPARQL complexity analysis, for the scope of this work we rely on the following qualitative definition:

Definition 1 (Heavy (light) SPARQL query). *Given a set Q of SPARQL queries, a query $q \in Q$ is defined as heavy (light) if it requires considerable (little) computational and memory resources.*

We extract SPARQL 1.1 features from logs with Apache Jena 3.1.1: we combine what is proposed in recent literature [5,7,18] and experiment with five configurations (Table 2). Besides disjunctive statements [7], *Group 1* includes the number of joins, of `FILTER`s, and of regular expressions used in filters. *Group 2* adds the count of triple patterns, while *Group 3* also includes the count of `*` operators in `SELECT` statements (more than one might be present in case of nested queries). *Group 4* adds a number of other features, including `FILTER`-specific operators, projections, grouping operations, pagination, and the count of basic graph patterns. Finally, the complete set of features (*All*) adds the *type* of joins included in a query. Besides the six join configurations described in [7], we also keep track of the count of completely unbound and bound statements. In this paper we do not consider property paths, or negations (`FILTER NOT EXISTS`). We choose two binary classifier models, support vector machine classification

Table 2. Feature vectors configurations in our experiments (`b`=bound, `u`=unbound. Examples:`?s foaf:name ?n` → `ubu`, `?s ?p?o` → `uuu`)

Group 1	• `UNION`s count	• `FILTER`s count	
	• `OPTIONAL`s count	• `FILTER`s `regex` count	
	• Joins count		
Group 2	• Triple patterns count		∪ Group 1
Group 3	• `SELECT *` count		∪ Group 2
Group 4	• `FILTER`s expressions count	• `REDUCED`s count	∪ Group 3
	• `FILTER`s `strstarts` count	• `BIND`s count	
	• `FILTER`s `contains` count	• Basic graph patterns count	
	• `FILTER`s `in` count	• `GRAPH`s count	
	• `FILTER`s `or` count	• `MINUS` count	
	• `FILTER`s `lang` count	• Presence of `OFFSET`/`LIMIT`	
	• Presence of `ORDER BY`	• `GROUP BY` variables count	
	• Projections count	• `VALUES` count	
	• Projection variables count		
All	• Join types count (`bub`,`bbb`,`uuu`,`ubu`,`ubb`,`bbu`,`uub`,`buu`)		∪ Group 4

(SVC) and multinomial Naïve Bayes (NB), and we train such models on the training set described above (results presented in Sect. 5).

4.3 Audience Metrics Extraction and Visitor Session Identification

Most traffic analytics metrics described in this group (Table 1) rely on the notion of visitor *session*, and require query session analysis, i.e. the analysis of groups of queries. It is therefore crucial to (i) provide a Linked Data-compliant definition of session, and (ii) detect such sessions with acceptable precision and recall.

There is no general consensus on the definition of visitor session on the web [14, 20]. Linked Data is no exception, as there seems to be no work in the Linked Data research community that tackles session detection identification. We adapt the definitions presented in [14] to Linked Data:

Definition 2 (Session). *A Session is defined as a sequence of requests issued with no significant interruptions by a uniquely identified visitor to a Linked Dataset. A session expires after a period of inactivity. Requests can either be HTTP lookups or SPARQL queries.*

Definition 3 (Visitor). *A visitor is a client uniquely identified by an IP address and a user agent string. Visitors can issue either HTTP operation or SPARQL queries.*

Besides problems shared with the traditional web (e.g. how to define a *period of inactivity* between sessions?), Linked Data session patterns might differ from traditional web sessions: Linked Data is meant for human consumption, *and* software agents: does this influence the shape of sessions? Does Linked Data dual protocol (HTTP lookups and SPARQL) lead to different patterns?

We adopted the variant of hierarchical agglomerative clustering (HAC) presented by Murray et al. in [14]. Although originally conceived for classic web logs analysis, we show that HAC has good performance also in a Linked Data scenario. HAC adopts a purely time-based gap detection, which is performed with hierarchical agglomerative cluster analysis. HAC is visitor-centered, and identifies sessions from the burstiness of a visitor's activity. HAC finds visitor-specific cut-off thresholds, and it does not need a global fixed cut-off threshold. For each visitor, HAC carries out two steps (Fig. 5a). First, it scans the gaps between HTTP requests to find a time interval that significantly increases the variance (Fig. 5a, line 3): such interval becomes the visitor-specific session cut-off. Then, it groups HTTP requests into sessions according to the cut-off (Fig. 5a, line 4). Figure 5b shows the cut-off threshold choice in detail: the algorithm loops through inter-session gaps in ascending order (lines 4–12); at each iteration it computes the time difference δ_g between the current gap g and the mean μ_I of the set of processed gaps I (line 6). If the ratio of δ_g and the standard deviation σ_I is greater than r_{max} (line 7), then r_{max} is replaced by such ratio and the cut-off c_v is set to g (lines 8,9). Figure 5c shows how sessions are built: if the gap between a request and the previous one is shorter than the cut-off c_v (line 4), the request is added to the current session (line 5). Otherwise, a new session is created (line 9).

```
Algorithm: HAC
Input   : The set $R = \{R_1, \ldots, R_n\}$ of requests for
          $n$ visitors
Output  : The sessions $S$ of all $n$ visitors
1 foreach visitor $v$ do
2     $G_v \leftarrow$ get inter-requests time gaps from $R_v$
3     $c_v \leftarrow$ SetCutOff ($G_v$)
4     $S_v \leftarrow$ BuildSessions ($R_v, c_v$)
5     add $S_v$ to $S$
6 end
```

(a)

```
Function: SetCutOff
Input   : The gap intervals $G_v$ for
          visitor $v$
Output  : The visitor-specific cut-off
          threshold $c_v$
1  $I \leftarrow \emptyset$
2  $r_{max} \leftarrow 0$
3  sort $G_v$ in ascending order
4  foreach gap $g \in G_v$ do
5      if $I \neq \emptyset$ then
6          $\delta_g \leftarrow (g - \mu_I)$
7          if $\delta_g / \sigma_I > r_{max}$ then
8              $r_{max} \leftarrow \delta_g / \sigma_I$
9              $c_v \leftarrow g$
10         end
11     end
12     $I \leftarrow I \cup \{g\}$
13 end
```

(b)

```
Function: BuildSessions
Input   : The requests $R_v$ for visitor $v$,
          the visitor-based threshold $c_v$
Output  : The collection of sessions $S_v$
          for visitor $v$
1  $s \leftarrow$ new session
2  foreach request $r_i \in R_v$ do
3      gap $g \leftarrow (t_{r_i} - t_{r_{i-1}})$
4      if $g < c_v$ then
5          add $r_i$ to $s$
6      end
7      else
8          $S_v \leftarrow S_v \cup \{s\}$
9          $s \leftarrow$ new session
10     end
11 end
```

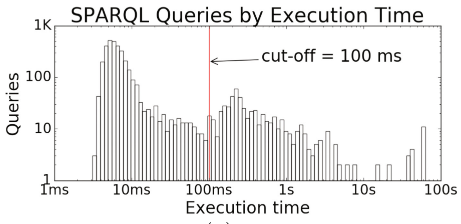
(c)

Fig. 5. HAC: (a) main function, (b) how per-visitor thresholds are set, (c) how HTTP/SPARQL requests are grouped in sessions.

5 Results

Heavy and Light SPARQL Queries. The light/heavy classifier must undergo ad-hoc training with dataset-specific access logs. To reproduce results, we describe how the classifier can be trained on public available USEWOD 2015 DBpedia 3.9 access logs [11] (we are not allowed to disclose the British National Bibliography access logs). The dataset used in this experiment is created as follows: we select random distinct queries from DBpedia 3.9 logs included in the USEWOD 2015 dataset. We execute such SPARQL queries on a local clone of DBpedia 3.9, and measure the response time of each query. Queries are issued from localhost. We run queries multiple times, to compute mean execution time μ_t and standard deviation ρ_t. We discard queries that led to HTTP and SPARQL errors, and queries with relative standard deviation $\rho_t/\mu_t > 0.8$. The resulting dataset contains 3,752 records (3,682 `SELECT`, 39 `CONSTRUCT`, and 31 `ASK`). We exclude `DESCRIBE` queries. We then choose a cut-off threshold $t_{cut-off}$=100ms, to separate *light* from *heavy* queries. Such value is chosen arbitrarily after manual inspection of the dataset: Fig. 6a shows the distribution of SPARQL against their execution time: the majority of queries falls within the first buckets. Note the extremely slow queries around 100 s. The chosen $t_{cut-off}$ leads to 3,192 *light* and 560 *heavy* queries (hence, the dataset is skewed towards *light* queries). Note that since we rely on a qualitative and scenario-dependent definition of *heavy* SPARQL query, the value of $t_{cut-off}$ can be replaced with another user-defined value, perhaps more meaningful under different conditions (e.g. different triplestore, server architecture, dataset). This paper only evaluates the quality of predictions with $t_{cut-off}$=100ms. Because supervised learning performance is influenced by $t_{cut-off}$, model re-training is required if such parameter is modified. We split the dataset in a training test and a test set. The two splits are generated with random stratified sampling, and account for 80% and 20% of records respectively. Training includes 2,553 *light* and 448 *heavy* queries; test includes 639 *light* and 112 *heavy* queries.

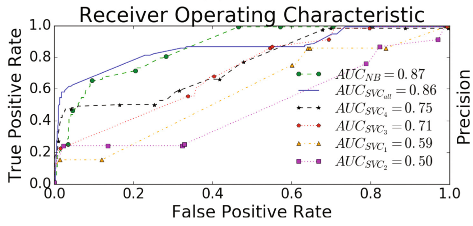
We run 10-fold cross-validated model selection with grid search over both support vector classifier and multinomial Naïve Bayes, using F1 score as scoring function: for SVC we iterate over $C = [0.1, 1, 10, 100]$ and linear versus radial basis function (RBF) kernels. For RBF kernels we try $\gamma = [0.1, 0.2, 0.5]$. Multinomial Naïve Bayes (NB) is trained with $\alpha = [0.5, 0.1, 0.2, 0.3, 0.4, 0.5, 1, 10]$, and learned class prior probabilities versus uniform prior. We fine-tune SVC model selection with randomized search over the ranges mentioned above, and find the best SVC estimator to have $C = 12.96$ and RBF kernel with $\gamma = 0.10$. The best NB classifier has $\alpha = 12.7$ and uniform prior. For SVC, we also train our models with each of the five groups of features described in Table 2. Figure 6b shows the quality of predictions by class when all features are used: although Naïve Bayes does not lead to reliable predictions of *heavy* queries (F1= 0.56), SVC reaches a F1 score of 0.66 ($P = 0.87$, but recall lags at $R = 0.54$, leaving space for future improvements). *Light* queries are labeled with F1= 0.95 by SVC and F1= 0.91 by Naïve Bayes. The average F1 score of SVC is 0.91, while NB yields to 0.86. Figures 6c and d show that AUC-ROC for SVC and Naïve Bayes are close



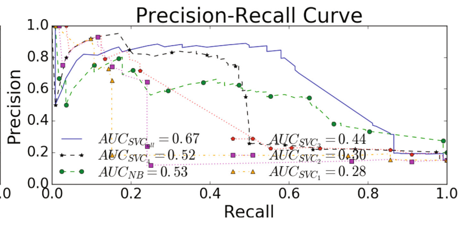
(a)

		P	R	$F1$	$F1_{avg}$	Acc
SVC_{all}	light	0.92	0.99	0.95	0.91	0.92
	Heavy	0.87	0.54	0.66		
NB_{all}	light	0.94	0.88	0.91	0.86	0.85
	Heavy	0.49	0.67	0.56		

(b)



(c)



(d)

Fig. 6. *Heavy/light* SPARQL classification on USEWOD 2015: (a) queries distribution by execution time; (b) breakdown of performance (all features); (c) ROC and (d) PR curve for each feature configuration in Table 2.

(0.87 and 0.86 respectively). Nevertheless, due to class imbalance, we consider more reliable the area of the precision-recall curve (AUC-PR), where SVC shows the best result (AUC-PR= 0.67). Figures 6c and d also show that the entire set of features `All` is more effective than `Groups 1-4` (Table 2).

Visitor Session Identification. We assess the quality of the results of the session detection algorithm. We measure how well the adopted algorithm detects the beginning of a session. We compare precision, recall, and F1 score of HAC with sessions with fixed cut-offs of 15, 30, and 60 min respectively (these are common fixed-length session durations in literature [1,14]).

Our evaluation dataset spans three consecutive random days of access records to the British National Bibliography Linked Dataset. The dataset includes 137 anonymised distinct visitors with at least 5 visits. Known search engine crawlers have been filtered out. Overall, the dataset includes 16,426 HTTP requests, 67.8% (11,128) of which have been issued by 10 distinct visitors that used software libraries, 31.6% (5,206) by 115 distinct visitors with desktop browsers, and 0.6% by other types of user agents (UA) - e.g. mobile browsers. Records include a timestamp, an HTTP (or SPARQL) request, and the user agent of the visitor. Records have been manually annotated by a domain expert as *session start* or *internal*: a total of 576 human-annotated session starts have been found, leading to 439 human-validated session gaps (the total count of session starts minus the 137 initial sessions for each visitor). As shown by Fig. 7a, session cut-offs are considerably spread out from their means μ. Data suggests that outliers greatly influence the means: software libraries are particularly affected with a standard deviation σ three times bigger than the mean. In this case, 50% of cut-offs are shorter than 29 m, but outliers bring the mean beyond 1 h.

Our tests show that HAC outperforms fixed-length cut-offs (Fig. 7b). When considering all user agents (*All UAs*), HAC's precision is always higher than the three fixed-length cut-offs (the 15 min cut-off leads to better recall, but precision is not satisfactory). Family-specific session detection shows that HAC is outperformed in the case of browsers (this presumably corresponds to human browsing sessions). Fixed cut-offs attempts show that precision increases with larger thresholds, but recall always decreases: this is because fixed-length sessions do not perform well when session gaps are highly spread out from the mean. Such behaviour is more prominent for software libraries than for desktop browsers, since session gaps of software libraries are more scattered: Fig. 7a shows that they have the highest coefficient of variation among the three categories ($c_v = \sigma/\mu = 2.57$), i.e. they have the highest relative standard deviation.

Impact of Mining Tasks and Lessons Learned. We show that our binary supervised classifier based solely on SPARQL syntactic features helps identifying workload peaks of SPARQL endpoints. This approach based on statistical learning overcomes missing query response time in access logs, as we cannot assume that logs include such information. Results show that SVC leads to encouraging results, with satisfactory detection of light queries. Heavy queries prediction comes with acceptable precision, although recall shows margin for improvement, namely through enhancements in feature extraction (e.g. adding property paths support) and training on larger datasets. The main disadvantage of our approach is the need for a adequately large and diverse training set. Class imbalance is also a limitation: *heavy* queries are rare - among more than 60k distinct queries from USEWOD logs which return `200 OK`, only 560 led to mean execution times above the chosen cut-off with satisfactory relative standard deviation. Another limitation of our approach is the need to re-train the model if $t_{cut-off}$ is changed.

Session detection proves vital to deliver a more accurate estimate of the traffic on a Linked Data server: sessions enable understanding the duration of each visit, how often visitors come back, and how many distinct resources are requested during such time frame. They also serve as a course-grained alternative to the request count, thus helping gauging visitor engagement. The main benefit of HAC is its performance with intertwined human-machine traffic - a distinctive feature of Linked Data: we show that HAC outperforms fixed cut-offs on session gaps with high relative standard deviation. This is important for session analysis in Linked Data because, unlike traditional web, we must deal at the same time with *both* visits from desktop browsers (presumably by humans, hence with more condensed session gaps - see Fig. 7a), and from software libraries (that generate more scattered session durations). HAC also supports the dual access-protocol nature of Linked Data, since the time-based version of HAC presented in this paper is access protocol-independent. The heuristic can therefore be adopted as baseline, thus laying the foundations for content-based session detection heuristics that take into account SPARQL, HTTP, or mixed sessions. Note that, since we do not inject tracking code to identify single visitors, session detection with HAC should be considered a lower-bound estimate: we cannot circumvent intermediate components between visitors and datasets - e.g. caches (ISP, browsers,

	μ	σ	c_v	Median	95th perc	Support
All UAs	4h16m	6h16m	1.47	1h35m	18h20m	439
Desktop Browsers	5h43m	6h52m	1.20	2h57m	19h46m	259
Software Libraries	1h14m	3h10m	2.57	29m	3h18m	158

(a)

		P	R	F1
All UAs	**HAC**	**0.91**	0.72	**0.80**
	15 m	0.58	**0.99**	0.73
	30 m	0.72	0.75	0.73
	60 m	0.87	0.67	0.75
Desktop Browsers	**HAC**	0.88	0.63	0.73
	15 m	0.60	**0.99**	0.75
	30 m	0.77	0.96	0.86
	60 m	**0.91**	0.87	**0.89**
Software Libraries	**HAC**	**0.96**	0.93	**0.95**
	15 m	0.53	**1.00**	0.69
	30 m	0.43	0.23	0.30
	60 m	0.76	0.15	0.26

(b)

Fig. 7. (a) Gaps between manual annotated sessions in the evaluation dataset, and (b) session detection evaluation results: HAC algorithm vs fixed cut-offs (*All UAs* also includes 22 cut-offs from UA types not included in the table).

or others), proxy servers, NAT. Besides, visitors might fake user agent strings, thus leading to imprecise visitor identification.

Insights on the British National Bibliography Traffic Logs. We tested the the system on 13 months of logs of the British National Bibliography dataset (March 2014–April 2015). The dataset contains almost 100 million triples about books and serials, and is accessible with HTTP and SPARQL. The platform automatically filtered out 99.4% of records, as such requests were generated by search engines and malicious crawlers. Only 0.6% are genuine calls to the triplestore. The platform shows that the combined traffic of HTTP operations and SPARQL queries over the observed period increased by 30%. Interestingly, we identified a 95-fold increase in requests from software libraries, i.e. clients that interact with the triplestore by means of SPARQL queries. We identified 49 unique applications of this kind. Desktop browsers generated 62% of requests (either HTTP operations or SPARQL queries). SPARQL access accounts for 29% of total requests. 6% of queries have been classified as *heavy*. We identified 37 days that show unusual traffic spikes. One of these findings consists in a 1-h spike of 10 thousands *light* `SELECT` SPARQL queries. The traffic analytics platform also indicates the town and host network of origin, and shows that such queries come from a Java-based application. With this in mind, dataset administrators quickly found in logs that such spike consists in thousands of identical queries - probably originated from a bug in the client application. The platform shows that 33% of requests come from the United States, with the United Kingdom totalling 22%. Nevertheless, the most common city of origin of visitors is Frankfurt, in Germany. WHOIS lookups shows that the dataset has been accessed by more than 250 universities, at least 100 government-related host networks worldwide, and at least 20 other libraries. Our platform also measures user retention: over the analysed 13 months, bounce rate is 48%, meaning that almost half of visitors never came back after the first visit. The average monthly percentage of new visitors is 74%. Visitor session detection shows that visits

Instance	Freq (%)
resource/009910399	2.4
resource/007073756	1.9
person/LewisCS%28CliveStaples%291898-1963	0.6
person/TolkienJRR%28JohnRonaldReuel%291892-1973	0.6
resource/007073756/publicationevent/LondonHarperCollins1993c1978	0.6

(a)

Class	Freq (%)
dcterms:BibliographicResource	0.7
bibo:Author	0.5
bibo:Book	0.4
resource/Author	0.2
foaf:Person	0.1

(b)

Property	Freq (%)
bibo:isbn10	8.7
dcterms:title	4.9
rdfs:label	3.2
dcterms:creator	2.4
foaf:name	2.3

(c)

Fig. 8. Top-5 RDF instances, classes, and properties of the BNB dataset (HTTP operations *and* SPARQL queries). (a) instances, (b) classes, and (c) properties.

generated by software libraries have bigger, deeper, and longer sessions: the average daily size of software libraries sessions is 24 resources (against 2 resources for sessions originated from desktop browsers). The average daily depth of software libraries sessions is 11 unique resources (desktop browsers show on average 2 distinct resources per session). Software library sessions last in average 1 h and 3 min, while desktop browser sessions only 27 min. Figure 8 shows the most popular RDF instances, classes, and properties of the dataset.

Reproducibility and Public Demo. A public demo of the system is available at http://bit.ly/ld-traffic [3]. Datasets used for experiments with the SPARQL classifier and the session detector are available at http://bit.ly/traffic-ESWC2017. The heavy/light SPARQL classifier is implemented with `scikit-learn` 0.18.1 `SVC` and `MultinomialNaiveBayes`. Hyperparameters not listed in Sect. 4 are set to scikit-learn 0.18.1 defaults. Pseudo-code of the session detection algorithm is presented in Fig. 5. All experiments have been executed on an Intel Xeon E5-2420 v2 @ 2.20 GHz (1 socket, 6 cores, 12 threads), 128 GB RAM, Ubuntu 14.04 LTS, Virtuoso version 7.10.3211 (Virtuoso memory usage settings: `NumberOfBuffers=5450000, MaxDirtyBuffers=4000000`).

6 Conclusions and Future Work

We present a novel traffic analytics platform that relieves publishers from manual and time-consuming access log mining. We add novel Linked Data-specific metrics - to break down traffic by RDF content, capture SPARQL insights, and properly interpret 303 patterns. Platform aside, we also propose two mining tasks adopted by the system: the reconstruction of Linked Data visitors sessions - which we are the first to achieve with time-based hierarchical agglomerative clustering (establishing a baseline for future Linked Data-optimized heuristics), and the detection of workload peaks of SPARQL endpoints, achieved by predicting heavy and light SPARQL queries with a novel approach based on supervised learning and SPARQL syntactic features. The analysis of 13 months of access logs of the British National Bibliography dataset shows that our system effectively reveals visitors insights otherwise hidden to dataset publishers. These findings are useful, among all, to gauge SPARQL traffic spikes, and monitor trends (e.g. HTTP vs SPARQL traffic over time). They also help justifying investment in Linked Data, and enhancing the popularity of a dataset: for example, the awareness of decreasing user retention might prompt for better promotion (e.g. hackatons, spreading the word on community mailing lists, etc.). Also, if portions of a dataset are never accessed, perhaps better data documentation is required.

Future extensions will include additional metrics, such as statistics about noisy traffic that is now simply discarded by the platform (i.e. web crawlers). The heavy/light classifier feature set will be refined, and we will investigate whether this approach generalizes to additional linked datasets. We will enhance time-based session detection with content-based heuristics, such as relatedness of subsequent SPARQL queries, and structure and type of requested RDF entities.

Acknowledgements. This work has been supported by the TOMOE project funded by Fujitsu Laboratories Limited in collaboration with Insight Centre at NUI Galway.

References

1. Arlitt, M.: Characterizing web user sessions. ACM SIGMETRICS Perform. Eval. Rev. **28**(2), 50–63 (2000)
2. Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbussche, P.-Y.: SPARQL web-querying infrastructure: Ready for action? In: Alani, H., Kagal, L., Fokoue, A., Groth, P., Biemann, C., Parreira, J.X., Aroyo, L., Noy, N., Welty, C., Janowicz, K. (eds.) ISWC 2013. LNCS, vol. 8219, pp. 277–293. Springer, Heidelberg (2013). doi:10.1007/978-3-642-41338-4_18
3. Costabello, L., Vandenbussche, P., Shukair, G., Deliot, C., Wilson, N.: Access logs don't lie: Towards traffic analytics for linked data publishers. In: Proceedings of ISWC Posters & Demos Track (2016)
4. Demartini, G., Enchev, I., Wylot, M., Gapany, J., Cudré-Mauroux, P.: BowlognaBench—Benchmarking RDF analytics. In: Aberer, K., Damiani, E., Dillon, T. (eds.) SIMPDA 2011. LNBIP, vol. 116, pp. 82–102. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34044-4_5
5. Dividino, R., Gröner, G.: Which of the following SPARQL queries are similar? why? In: Proceedings of LD4IE Workshop (2013)
6. Fasel, D., Zumstein, D.: A fuzzy data warehouse approach for web analytics. In: Lytras, M.D., Damiani, E., Carroll, J.M., Tennyson, R.D., Avison, D., Naeve, A., Dale, A., Lefrere, P., Tan, F., Sipior, J., Vossen, G. (eds.) WSKS 2009. LNCS (LNAI), vol. 5736, pp. 276–285. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04754-1_29
7. Gallego, M.A., Fernández, J.D., Martínez-Prieto, M.A., de la Fuente, P.: An empirical study of real-world SPARQL queries. In: Proceedings of USEWOD (2011)
8. Halfaker, A., Keyes, O., Kluver, D., Thebault-Spieker, J., Nguyen, T., Shores, K., Uduwage, A., Warncke-Wang, M.: User session identification based on strong regularities in inter-activity time. In: Proceedings of WWW, pp. 410–418 (2015)

9. Hasan, R., Gandon, F.: A machine learning approach to SPARQL query performance prediction. In: Proceedings of WI, vol. 1, pp. 266–273. IEEE (2014)
10. Heath, T., Bizer, C.: Linked Data: Evolving the Web into a Global Data Space. Synthesis Lectures on the Semantic Web. Morgan & Claypool, Palo Alto (2011)
11. Luczak-Roesch, M., Berendt, B., Hollink, L.: USEWOD 2015 Research Dataset (2015). http://dx.doi.org/10.5258/SOTON/379407
12. Mehrzadi, D., Feitelson, D.G.: On extracting session data from activity logs. In: Proceedings of ISS, p. 3. ACM (2012)
13. Möller, K., Hausenblas, K., Cyganiak, R., Handschuh, S.: Learning from linked open data usage: Patterns & metrics. In: Proceedings of Web Science (2010)
14. Murray, G.C., Lin, J., Chowdhury, A.: Identification of user sessions with hierarchical agglomerative clustering. In: ASIS&T, vol. 43(1), 1–9 (2006)
15. Pallis, G., Angelis, L., Vakali, A.: Model-based cluster analysis for web users sessions. In: Hacid, M.-S., Murray, N.V., Raś, Z.W., Tsumoto, S. (eds.) ISMIS 2005. LNCS (LNAI), vol. 3488, pp. 219–227. Springer, Heidelberg (2005). doi:10.1007/11425274_23
16. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 30–43. Springer, Heidelberg (2006). doi:10.1007/11926078_3
17. Petridou, S.G., Koutsonikola, V.A., Vakali, A.I., Papadimitriou, G.I.: Time-aware web users' clustering. IEEE Trans. Knowl. Data Eng. **20**(5), 653–667 (2008)
18. Picalausa, F., Vansummeren, S.: What are real SPARQL queries like? In: Proceedings of SWIM, p. 7. ACM (2011)
19. Schmidt, M., Meier, M., Lausen, G.: Foundations of SPARQL query optimization. In: ICDT, pp. 4–33. ACM (2010)
20. Ye, C., Wilson, M.L., Rodden, T.: Develop, implement, and improve a web session detection model. In: Proceedings of IIiX, pp. 336–338. ACM (2014)