CEUR-WS.org/Vol-2324/Paper18-PMarcel.pdf

Qualitative Analysis of the SQLShare Workload for Session Segmentation

Veronika Peralta, Willeme Verdeaux, Yann Raimont, Patrick Marcel
University of Tours
Blois, France
veronika.peralta|patrick.marcel@univ-tours.fr,willeme.verdeaux|yann.raimont@etu.univ-tours.fr

ABSTRACT

This paper presents an ongoing work aiming at better understanding the workload of SQLShare [9]. SQLShare is database-as-a-service platform targeting scientists and data scientists with minimal database experience, whose workload was made available to the research community. According to the authors of [9], this workload is the only one containing primarily ad-hoc hand-written queries over user-uploaded datasets. We analyzed this workload by extracting features that characterize SQL queries and we show how to use these features to separate sequences of SQL queries into meaningful sessions. We ran a few test over various query workloads to validate empirically our approach.

1 INTRODUCTION

Analyzing a database workload offers many practical interests, from the monitoring of database physical access structures [2] to the generation of user-tailored collaborative query recommendations for interactive exploration [6]. There has been much attention lately devoted to the analysis of user past activities to support Interactive Database Exploration [8]. OLAP analysis of data cubes is a particular case of IDE, that takes advantage of simple primitives like drill-down or slice-and-dice for the navigation of multidimensional data. These particularities enable the design of approaches for characterizing user explorations in how focus they are [4], in how contributive a query is to the exploration [3], or even in how to ensure that a sequence of analytical queries forms a coherent exploration [13].

Transposing these works to regular, non multidimensional SQL workloads raises many challenges. Even if a sequence of SQL queries is issued to explore the database content, non multidimensional relational schemas do not have regularities one expects from the multidimensional model, explorations may not be expressed through roll-up or drill-down operations, SQL queries may deviate from the traditional star-join pattern commonly used for analytical purpose, etc.

In this paper, we present a preliminary approach for analyzing SQL workloads, concentrating on the SQLShare workload of hand-written[1] queries over user-uploaded datasets. This workload includes raw sequences of queries made by some users, without further information on their intention. One of our objectives is to investigate whether this workload contains actual exploration activities, and more particularly how to extract such explorations. In what follows, we consider that an exploration is a coherent sequence of hand-written queries, that all share the same goal of fulfilling a user's information need that may not be well defined initially. Identifying such activities has several applications in supporting interactive database exploration (IDE), like understanding users' information needs, identifying struggling during the exploration, providing better query recommendations, etc. This is important since, usually, systems used for IDE do not offer such facilities.

To identify explorations from a SQL workload, we use a technique first proposed in [3] to score the quality of OLAP explorations. This technique consists of characterizing a query by a set of simple features that are intrinsic to a query or that relate the query to its neighbor in the sequence. While in [3] this technique of feature extraction was used with supervised machine learning to score the quality of OLAP explorations, in the present work we use these features to partition an SQL workload into coherent explorations.

The paper is organized as follows. The next section discusses related work. Section 3 presents our model of queries, tailored for SQL queries. Section 4 details the features considered and how they are extracted. Section 5 introduces our segmentation strategy and Section 6 reports the results of the tests we conducted. Section 7 concludes and draws perspectives.

2 RELATED WORK

In this section we present related work concerning real SQL workloads and workload analysis.

2.1 Real SQL workloads

SQLShare. The SQLShare workload is the result of a Multi-Year SQL-as-a-Service Experiment [9], allowing any user with minimal database experience to upload their datasets on-line and manipulate them via SQL queries. What the authors wanted to prove with this experiment is that SQL is beneficial for data scientists. They observed that most of the time people use scripts to modify or visualize their datasets instead of using the SQL paradigm. Indeed, most user needs may be satisfied by first-order queries, that are much simpler than a script, but have the initial cost of creating a schema, importing the data and so on. SQL-as-a-Service frees the user of all this prior work with a relaxed SQL version.

The SQLShare workload is composed of 11,137 SQL statements, 57 users and 3,336 user's datasets. To the best of our knowledge, as reported by the authors of [9], this workload is the only one containing primarily ad-hoc hand-written queries over user-uploaded datasets. As indicated in the introduction, hand-written means that the query is introduced manually by a human user, which reflects genuine interactive human activity over a dataset, with consideration between two consecutive queries.

The SQLShare workload is analyzed in [9], particularly to verify the following assumption:

[1]Consistently with the authors of [9], we use the term hand-written to mean, in this context, that the query is introduced manually by a human user, which reflects genuine interactive human activity over a dataset, with consideration between two consecutive queries.

© 2019 Copyright held by the author(s). Published in the Workshop Proceedings of the EDBT/ICDT 2019 Joint Conference (March 26, 2019, Lisboa, Portugal):

"We hypothesized that SQLShare users would write queries that are more complex individually and more diverse as a set, making the corpus more useful for designing new systems."

The authors indeed showed empirically that the queries in the SQLShare workload are complex and diverse. They also analyzed the churn rate of SQLShare users and conclude that most users exhibit a behavior that suggest an exploratory workload.

To our knowledge, and again as reported by the authors of [9], this workload is one of the two workloads publicly available to the research community, the other being the SDSS workload.

SDSS workload. SkyServer is an Internet portal to the Sloan Digital Sky Survey Catalog Archive Server; its Web and SQL logs are public [14]. The SQL log was produced by a live SQL database supporting both ad hoc hand-authored queries as well as queries generated from a point-and-click GUI. Many queries in the SDSS are actually not hand-written; they were generated by applications such as the Google Earth plugin or the query composer from the SkyServer website. Their cleaning and normalization took several months effort.

Sessions in this log were detected using heuristics:

"We arbitrarily start a new session when the previous page view from that IP address is more than 30 minutes old, i.e., a think-time larger than 30 minutes starts a new session. [...] Wong and Singh [1] chose the same 30 minute cutoff and we are told that MSN and Google use a similar heuristic."

The authors of [14] also acknowledge the difficulty of extracting human sessions from all those collected:

"We failed to find clear ways to segment user populations. We were able to ignore the traffic that was administrative or was eye-candy, leaving us with a set of 65M page views and 16M SQL queries. We organized these requests into about 3M sessions, about half of which were from spiders. The residue of 1.5M sessions had 51M page views and 16M SQL queries – still a very substantial corpus. [...] Interactive human users were 51% of the sessions, 41% of the Web traffic and 10% of the SQL traffic. We cannot be sure of those numbers because we did not find a very reliable way of classifying bots vs mortals."

Bots are programs that automatically crawled the SDSS and launch SQL queries. Such traffic cannot be classified as proper interactive data exploration with human consideration.

In [9], the authors compared the SQLShare workload and that of the SDSS, and conclude:

"SQLShare queries on average tend to be more complex and more diverse than those of a conventional database workload generated from a comparable science domain: the Sloan Digital Sky Survey (SDSS)."

Smaller SQL datasets. We are aware of other available SQL workloads. For instance, Kul et al. [11] analyze three specific query sets. The first one, Student assignments gathered by IIT Bombay, is made of a few hundreds queries answering homework assignments. The second dataset, publicly available, consists of around 200 queries gathered over 2 years from student exams at University of Buffalo. The third dataset consists of SQL logs that capture all database activities of 11 Android phones for a period of one month. The log consists of 1,352,202 SELECT statements that, being generated by an application, correspond to only 135 distinct query strings.

2.2 Workload analysis

Other scientific domains close to Database, like Information Retrieval or Web Search, have a long tradition of log analysis aiming at facilitating the searcher's task [17]. Many works extract features from queries or search sessions and use them to disambiguate the session's goal, to generate recommendations, to detect struggling in sessions, etc. Since databases tend to be more used in an exploratory or analysis fashion, as evidenced by the SQLShare workload, it is not a surprise that many recent works pay attention to the analysis of database workloads, in addition to those works analyzing workload for optimization or self-tuning purposes. We present some recent advances in this area, differentiating by the type of logs (OLAP logs and SQL logs).

Analyzing and detecting OLAP explorations. Logs of OLAP analyses are simpler than SQL ones in the sense that they feature multidimensional queries that can easily be interpreted in terms of OLAP primitives (roll-up, drill-down, slice-and-dice, etc.). In one of our previous works [13], we proposed an approach for detecting OLAP analyses phrased in SQL, by converting SQL queries into OLAP queries and then checking if two consecutive queries are sufficiently close in terms of OLAP operations. In our more recent work, we used supervised learning to identify a set of query features allowing to characterize focus zones in OLAP explorations [4], or to identify queries that better contribute to an exploration [3]. The present work can be seen as a continuation of those previous works, since we have the same objective as [13] and use the same technique as [3]. The main differences with these previous works is that we make no assumption about the type of queries in the workload (particularly, they may not be multidimensional queries), and we have no ground truth (i.e., no human manual inspection of each query) on the workload.

Analyzing SQL log. SQL workload analysis has recently attracted attention beyond query optimization, for instance for query recommendation [6] query autocompletion [10], or user interest discovery [12]. All these works use the SDSS workload for their tests. Embedded SQL code is analyzed in [15] to measure its quality, mainly for maintainability purpose. The authors quantify quality based on the number of operators (joins, unions), operands (tables, subqueries) and variables in the SQL code, experimenting with SQL codes embedded in PL/SQL, COBOL and Visual Basic. Jain et al. ran a number of tests on the SQLShare workload [9], some of them being reported above, showing the diversity and complexity of the workload. In [16], Vashistha and Jain analyze the complexity of queries in the SQLShare workload, in terms of the following query features: number of tables, number of columns, query length in characters, numbers of operators (Scan, Join, Filter), number of comparison operators (LE, LIKE, GT, OR, AND, Count), and the query run-time. They define two complexity metrics from these features: the Halstead measure (traditionally used to measure programs complexity) and a linear combination whose weights are learned using regression. Finally, a recent work investigated various similarity metrics over SQL queries, aiming at clustering queries [11] for better workload understanding. The authors run their tests on smaller SQL sets, as indicated above.

To our knowledge, our work is the first to propose an approach for segmenting hand-written SQL queries into meaningful sessions.

3 PRELIMINARIES

This section introduces the SQLShare workload and describes our hypothesis and preprocessing.

3.1 SQLShare workload preprocessing

From the 11,137 SQL statements we kept 10,668 corresponding to SELECT statements. The remaining statements (mainly updates, inserts and deletes) were filtered.

We implemented a preliminary session segmentation following a simple heuristic: keeping together the sequences of consecutive queries of a same user. As a result of the initial segmentation we obtained 451 sessions, counting between 1 and 937 queries (average of 23.65 queries per session, standard deviation of 75.05 queries). Furthermore, we made the initial hypothesis that queries appear in chronological order in the SQLShare workload. We noted that the queries of the workload do not come with timestamps, and we contacted the authors of the original SQLShare paper [9] who confirmed that the query order in the workload may not reflect the order in which queries were launched. Therefore, the disparate distribution of queries along sessions, in addition to some extremely long sessions, possibly disordered, calls for a smarter way of segmenting sessions.

3.2 Query and session abstractions

In what follows, we use the term *query* to denote the text of an SQL SELECT statement. We represent a query as a collection of fragments extracted from the query text, namely, projections, selections, aggregations and tables. These fragments abstract the most descriptive parts of a SQL query, and are the most used in the literature (see e.g., [6, 10]). But note that we do not restrict to SPJ (selection-projection-join) queries. Indeed, we consider all queries in the SQLShare workload, some of them containing arbitrarily complex chains of sub-queries.

Definition 3.1 (Query). A *query* over database schema DB is a quadruple $q = \langle P, S, A, T \rangle$ where:

(1) P is a set of expressions (attributes or calculated expressions) appearing in the main SELECT clause (i.e. the outermost projection). We deal with * wild card by replacing it by the list of attributes it references.
(2) S is a set of atomic Boolean predicates, whose combination (conjunction, disjunction, etc.) defines the WHERE and HAVING clauses appearing in the query. We considered indistinctly all predicates appearing in the outermost statements as well as in inner sub-queries.
(3) A is a set of aggregation expressions appearing in the main SELECT clause (i.e. the outermost projection). Including aggregation expressions appearing only in the HAVING clause is part of our future work.
(4) T is a set of tables appearing in FROM clauses (outermost statement and inner sub-queries). Views, sub-queries and other expressions appearing in FROM clauses are parsed in order to obtain the referenced tables.

Note that although we consider tables and selections occurring in inner sub-queries, we limit to the outermost queries for projections and aggregations, as they correspond to attributes actually visualized by the user. We intentionally remain independent of presentation and optimization aspects, specially the order in which attributes are projected (and visualized by the user), the order in which tables are joined, etc.

Finally, a *session* is a sequence of queries of a user over a given database.

Definition 3.2 (Session). Let DB be a database schema. A session $s = \langle q_1, \ldots, q_p \rangle$ over DB is a sequence of queries over DB. We note $q \in s$ if a query q appears in the session s, and $session(q)$ to refer to the session where q appears.

4 FEATURE EXTRACTION

In this section, we define a set of features to quantitatively describe different aspects of a SQL query and its context. We then describe the extraction procedure and the obtained scores.

4.1 Feature description

For each query, we extract a set of simple features computed from the query text and its relationship with other queries in a session. We intend to cover various aspects of a query in order to support different types of analysis and modeling based on query features.

The set of features is inspired from our previous work [3, 4], which models OLAP queries as a set of features capturing typical OLAP navigation.

From now on, to remove any ambiguity, we use the term metric to denote the functions that score query quality. The term feature is reserved to denote the score output by the function.

For the sake of presentation, we categorize metrics as follows: i) intrinsic metrics, i.e., only related to the query itself, ii) relative metrics, i.e., also related to the query's predecessor in the session, and iii) contextual metrics, i.e., related to the whole session, providing more context to the metrics. Table 1 presents an overview of metrics.

	Intrinsic metrics
NoP	Number of projections (attributs and expressions)
NoS	Number of selections (filtering predicates)
NoA	Number of aggregations, when there is a group-by clause
NoT	Number of tables
	Relative metrics
NCP	Number of common projections, with previous query
NCS	Number of common selections, with previous query
NCA	Number of common aggregations, with previous query
NCT	Number of common tables, with previous query
RED	Relative edit distance (effort to express a query starting from the previous one)
JI	Jaccard index of common query parts, with previous query
	Contextual metrics
NoQ	Number of queries (absolute position in the session)

Table 1: metrics for SQL queries

For all definitions given in this section, let $q_k = \langle P_k, S_k, A_k, T_k \rangle$ be the query occurring at position k in the session s over the instance I of schema DB. All the queries we considered are supposed to be well formed, and so we do not deal with query errors.

For the moment, we only considered metrics based on query text. Further metrics may be defined if the database instance is taken into account (for example, number of tuples in query result, precision and recall of query result w.r.t. previous query, execution time, etc.). But the computation of such metrics implies the execution of every query in the SQLShare dataset, which is not always available for confidentiality reasons (i.e. users did not agree to share their data), and thus considerably reduces the set of queries. We left such studies to future work.

4.1.1 *Intrinsic metrics.*

Intrinsic metrics are those that can be computed only considering the query q_k, independently of the session s and other queries in s. In other words, these metrics will give the same score to q_k, independently of s.

Number of Projections. $NoP(q_k)$ represents the number of projections (attributes and expressions) that are projected by the user. Expressions projected in inner sub-queries, but not projected by the outer query, are not considered as they do not appear in the query result.

$$NoP(q_k) = card(P_k) \tag{1}$$

Number of Selections. $NoS(q_k)$ represents the number of selections (elementary Boolean predicates) that appear in the query text, both in outer and inner sub-queries.

$$NoS(q_k) = card(S_k) \tag{2}$$

Number of Aggregations. $NoP(q_k)$ represents the number of aggregation expressions that are projected by the user. As for projections, expressions appearing in inner sub-queries are not considered.

$$NoA(q_k) = card(A_k) \tag{3}$$

Number of Tables. $NoP(q_k)$ represents the number of tables appearing in query text, both considering outer and inner sub-queries.

$$NoT(q_k) = card(T_k) \tag{4}$$

4.1.2 *Relative metrics.*

Relative metrics are those that are computed comparing the query q_k to the previous query in the session s. Let $q_{k-1} = \langle P_{k-1}, S_{k-1}, A_{k-1}, T_{k-1} \rangle$ be the previous query, being undefined for the first query of s (i.e., q_1). Each metric provides a default score for this limit case.

Number of Common Projections. $NCP(q_k, q_{k-1})$ counts the number of common projections of q_k relatively to q_{k-1}. The default value is 0 for q_1.

$$NCP(q_k, q_{k-1}) = \begin{cases} card(P_k \cap P_{k-1}) & \text{if } k > 1 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Number of Common Selections. $NCS(q_k, q_{k-1})$ counts the number of common selections of q_k relatively to q_{k-1}. The default value is 0 for q_1.

$$NCS(q_k, q_{k-1}) = \begin{cases} card(S_k \cap S_{k-1}) & \text{if } k > 1 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Number of Common Aggregations. $NCA(q_k, q_{k-1})$ counts the number of common aggregations of q_k relatively to q_{k-1}. The default value is 0 for q_1.

$$NCA(q_k, q_{k-1}) = \begin{cases} card(A_k \cap A_{k-1}) & \text{if } k > 1 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Number of Common Tables. $NCT(q_k, q_{k-1})$ counts the number of common tables of q_k relatively to q_{k-1}. The default value is 0 for q_1.

$$NCT(q_k, q_{k-1}) = \begin{cases} card(T_k \cap T_{k-1}) & \text{if } k > 1 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Relative Edit Distance. $RED(q_k, q_{k-1})$ represents the edition effort, for a user, to express the current query starting from the previous one. It is strongly related to query parts, and computed as the minimum number of atomic operations between queries, by considering the operations of adding/removing a projection, selection, aggregation or table. The considered cost for each observed difference (adding/removing) is the same.

$$\begin{aligned} RED(q_k, q_{k-1}) = {} & card(P_k - P_{k-1}) + card(P_{k-1} - P_k) \\ & + card(S_k - S_{k-1}) + card(S_{k-1} - S_k) \\ & + card(A_k - A_{k-1}) + card(A_{k-1} - A_k) \\ & + card(T_k - T_{k-1}) + card(T_{k-1} - T_k) \end{aligned} \tag{9}$$

For the limit case, we consider an empty query, $q_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$. Then, $RED(q_1, q_0) = card(P_1) + card(S_1) + card(A_1) + card(T_1)$.

Jaccard Index. $JI(q_k, q_{k-1})$ represents the ratio between the common query parts (projections, selections, aggregations and tables) and the union of query parts.

$$JI(q_k, q_{k-1}) = \frac{card(queryParts(q_k) \cap queryParts(q_{k-1}))}{card(queryParts(q_k) \cup queryParts(q_{k-1}))} \tag{10}$$

where $queryParts(q_k) = P_k \cup S_k \cup A_k \cup T_k$.

For the limit case, we consider an empty query, $q_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$. Then, $JI(q_1, q_0) = 0$.

4.1.3 *Contextual metrics.*

Contextual metrics are session dependent and make sense only in the context of a session. The same query q_k occurring in different sessions may be given different scores for metrics in this category. For now we consider only one such metric.

Number of Queries. $NoQ(q_k, s)$ counts the absolute position of query q_k in s.

$$NoQ(q_k, s) = k \tag{11}$$

4.2 Extraction protocol

This section briefly describes the procedure for extracting query features. We proceed in 3 steps:

(1) Filter the SQL workload in order to keep only SELECT statements, i.e. discard updates, inserts, etc.
(2) Extract query fragments (sets of projections, selections, aggregations and tables) for each query.
(3) Compute query features from query fragments.

For the first and second step we developed a C# script using the MSDN TSQL Parser. Removing all the non SELECT statements was straightforward. However, extracting query fragments required to deal with several particular cases. The major challenge was extracting projections. First, for getting the projections visualized by the user, we need to detect the outermost SELECT clause of a query and then extract the SELECT elements. When there is a star element (i.e : *SELECT * FROM T*) in the query, our script reaches the outermost FROM clause (looking for *T*). When *T* is a table, the script accesses schema metadata and obtains all the table attributes. If there is one or more sub-queries in the FROM clause, it repeats the previously described pattern until it finds either a table or a set of SELECT elements. Views and

WITH clauses were treated in a similar way. For now, we do not take into account the queries having more than one '*' in their SELECT elements. (i.e : *SELECT t1.*,t2.*,a,b FROM t1,t2*).

Note that this procedure for resolving star elements relies in the existence of schema metadata, i.e., having access to the corresponding datasets in order to obtain the list of attributes. However, some datasets are referenced in queries but are not present in the SQLShare released data because the user decided not to share them. For such queries (near 18% of the query workload), we could not resolve the list of projections and we needed to estimate the number of projections during third step (computing features).

The aggregations were obtained with a Parser's function that detects all the function calls within a query. The selections are all the atomic Boolean expressions contained in the queries and their subqueries. At this stage of our work we do not deal with predicate containment.

The third step computes query features as described in Equations 1 to 11. For the unresolved star elements, we estimated both the number of projections and the number of common projections, by taking advantage of the other queries in the exploration that list attributes of the "starred" tables. We used linear regression to estimate the remaining cases, with the AUTO-SKLEARN Python module [7], which is a module aiming at automatically choosing and parametrizing a machine learning algorithm for a given dataset, at a given cost (i.e., the time it takes to test different algorithms).

More precisely, we computed two distinct regressions, one for NoP and another one for NoCP. The methodology is the same for both and consist of the following:

(1) Each query are represented by 23 features : NoS, NoA, NoT, NCS, NCA, NCT, NoQ (neither NoP nor NCP since they are the target of the regression), the min, max, average and standard deviation of the NoP, NoS, NoA, and NoT, grouped by the exploration the query belongs to.
(2) The queries where NoP=0 (and consequently NCP=0) are removed from the data set.
(3) The data set is then split in 80/20 for cross-validation, and the AUTO-SKLEARN regression mode is used to fit the best regression. For the first regression, NoP is the target, for the second, NCP is the target. The maximum time to find a model is set to 180 seconds and the score used to measure the accuracy of the regression is R^2 (coefficient of determination) regression score function.
(4) The R^2 score of each regression is used to predict NoP (respectively NCP) for the queries removed at step 2.
(5) Degenerated cases (e.g., a predicted number of common projection that is greater than the predicted number of projections) are handled manually.

4.3 Analysis of query features

Table 2 summarizes the results of feature extraction. Value distributions are shown in Figure 1.

A first remark is that many queries have a high number of projections. Indeed, 38 queries (out of 10,668) project more than 100 expressions, while more than 21% project more than 10 expressions. The number of common projections is also high. The number of the other query fragments is less impressive. Less than 1% of queries exceed 10 selections, 10 aggregations or 10 tables. Average and standard deviation confirm this disproportion,

Feature	Min	Max	Avg	StdDev	Median	75pc	90pc
Intrinsic metrics							
NoP	1	509	9.36	22.51	5	10	18
NoS	0	83	1.19	3.10	1	1	3
NoA	0	49	0.41	2.02	0	0	1
NoT	0	84	1.50	3.29	1	1	2
Relative metrics							
NCP	0	509	4.90	17.58	1	5	12
NCS	0	82	0.59	1.96	0	1	2
NCA	0	48	0.21	1.11	0	0	1
NCT	0	83	0.85	2.05	1	1	2
RED	0	1020	11.32	27.21	4	12	25
JI	0	1	0.45	0.39	0.43	0.83	1
Contextual metrics							
NoQ	1	937	23.65	75.05	4	13.5	50

Table 2: Range, average, standard deviation, median and two percentiles for query features on the SQLShare dataset

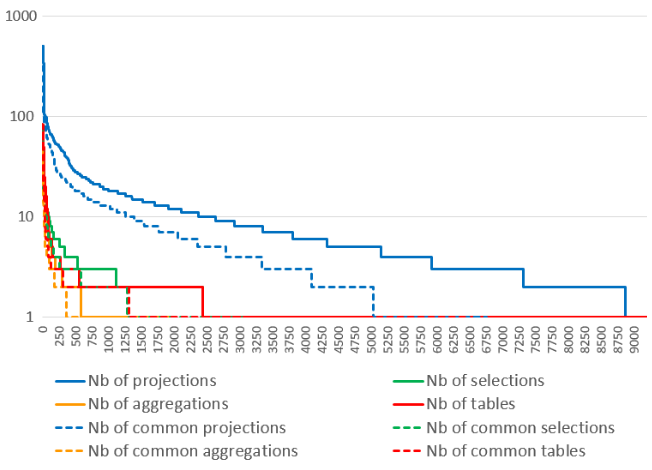

Figure 1: Value distribution of query features in the SQLShare dataset.

while median values show that most queries have few projections, selections, aggregations and tables.

Focusing in longer queries (with more query fragments), at 90-percentile, queries have 18 projections, 3 selections, 1 aggregation and 2 tables, while at 75-percentile those values are 10, 1, 0 and 1 respectively. Indeed, as expected, there is a large number of short queries (having less fragments): 82% of queries have no aggregations and 44% have no selections, while 20% have a unique projection and 78% have a unique table. Interestingly, 6% of queries have no table in the FROM clause. An example of such queries is "SELECT 1+2".

Concerning common fragments between contiguous queries, almost half of the queries have 1 common projection and 1 common table but no common selections nor aggregations, while there is more sharing at 75-percentile. The remaining two metrics, Relative Edit Distance and Jaccard Index, inform about the combination of such common fragments. Specifically, half of the queries differ in 4 or less fragments (RED=4) and have at least 43% of its fragments in common (JI=0.43), w.r.t. previous query.

Furthermore, only 29% of queries have nothing in common with the previous query in their sessions (JI=0).

The next section discusses how to take into account these features for fragmenting sessions.

5 SESSION SEGMENTATION

The previous section presented various statistics about queries in the SQLShare workload. A preliminary session segmentation (contiguous queries of a same user) resulted in some extremely long sessions (maximum of 937 queries) with 29% of queries having nothing in common with their immediate predecessor. In this section, we explore how to segment sessions in a smarter way.

Session segmentation has been previously studied for the SDSS workload [14]. In their study, the authors consider that a new session starts after 30 minutes of think-time (time spent between two queries). A similar problem was largely studied for the segmentation of web traces (see for example [18]) proposing the same 30-minutes cutoff. Search engine providers, like MSN and Google, use similar heuristics. Contrarily to those works, the published SQLShare workload does not include query timestamps. We need to explore other heuristics for session segmentation.

Intuitively, our idea is to compare contiguous queries in a session and segment when queries are dissimilar enough. Based on query features described in the previous section, we investigate 5 similarity indexes:

Edit Index. It is based on the Relative Edit Distance (RED) query feature, normalizing it with respect to an arbitrary threshold, set to 10 operations.

$$EditIndex(q_k) = max\{0, 1 - \frac{RED(q_k, q_{k-1})}{10}\} \quad (12)$$

Jaccard Index. It is the Jaccard Index defined in Equation 10, which is normalized by definition.

Cosine Index. It is calculated as the Cosine of vectors consisting in first 8 query features, namely, NoP, NoS, NoA, NoT, NCP, NCS, NCA and NCT. Let $x = \langle x_1, \ldots, x_8 \rangle$ and $y = \langle y_1, \ldots, y_8 \rangle$ be the vectors for queries q_k and q_{k-1} respectively.

$$CosIndex(q_k, q_{k-1}) = \frac{\sum x_i . y_i}{\sqrt{\sum x_i^2 . \sum y_i^2}} \quad (13)$$

Common Fragments Index. It is calculated as the number of common fragments, normalizing it with respect to an arbitrary threshold, set to 10 fragments.

$$CFIndex(q_k, q_{k-1}) = min\{1, \frac{NCF}{10}\} \quad (14)$$

where $NCF = NCP(q_k, q_{k-1}) + NCS(q_k, q_{k-1}) + NCA(q_k, q_{k-1}) + NCT(q_k, q_{k-1})$

Common Tables Index. It is calculated as the number of common tables, normalizing by the highest number of tables in the session.

$$CTIndex(q_k, q_{k-1}) = \frac{NCT(q_k, q_{k-1})}{max\{NoT(q) | q \in session(q_k)\}} \quad (15)$$

Note that these indexes calculate complementary aspects of query similarity. Edit Index and Common Fragment Index count differences (resp., common fragments) as absolute values (normalized with a given threshold). Jaccard Index is a compromise of the previous ones, computing the ratio of common fragments. Cosine Index is computed using features (the value of the metrics) instead of comparing sets of fragments; it captures the variability in query complexity. And finally, Common Table Index responds to the intuition that common tables have more impact than the other common fragments, and it is normalized with respect to the number of tables used in the user session.

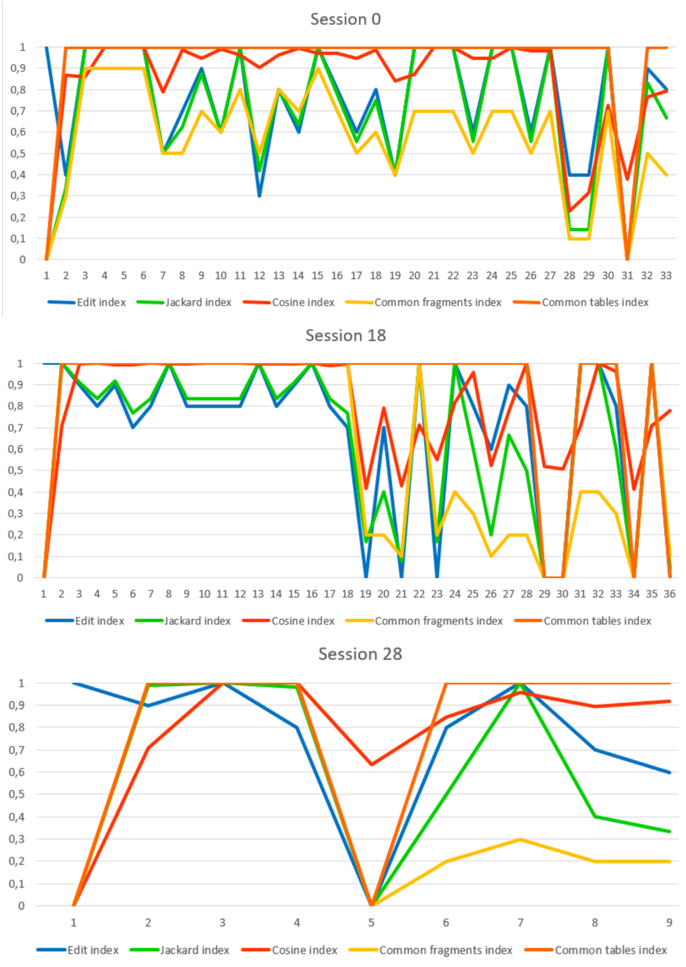

Figure 2: Comparison of similarity metrics for 3 sessions.

Figure 2 depicts the similarity metrics for 3 sessions of different sizes. Looking at Session 28, the shorter one, it seems quite clear that the session may be split in two parts, by cutting between queries 4 and 5. All similarity metrics agreed. Things are less evident for Session 0. One split seems evident (at query 31), but some others may be discussed (e.g. at queries 28, 29 and 12). Decision depends on thresholds. Finally, Session 18 presents a first part, with a focused analysis, via similar queries, and a second part, more exploratory, with varied queries. Even if indexes do not always agree, their majority seems to indicate a tendency.

6 EXPERIMENTS

In this section we report the results of the experiments conducted to validate our segmentation approach. We first experiment with the SQLShare dataset. We then experiment with other datasets for which a ground truth is available, in order to better underline the effectiveness of our approach.

	Before segmentation					After segmentation				
	Min	**1 quartile**	**2 quartile**	**3 quartile**	**Max**	**Min**	**1 quartile**	**2 quartile**	**3 quartile**	**Max**
Nb queries	1.00	2.00	6.00	19.75	936.00	1.00	1.00	3.00	6.00	78.00
Avg NCP	0.00	1.00	2.40	5.20	305.60	0.00	1.00	3.00	7.00	509.00
Avg NCS	0.00	0.00	0.17	0.72	7.31	0.00	0.00	0.00	1.00	82.00
Avg NCA	0.00	0.00	0.00	0.18	4.00	0.00	0.00	0.00	0.00	48.00
Avg NCT	0.00	0.66	0.97	1.00	4.00	0.00	0.80	1.00	1.00	83.00
Avg NCF	0.00	2.00	4.20	7.33	306.33	1.00	2.69	5.00	9.50	510.00
Avg RED	0.00	2.30	4.64	8.29	204.73	0.00	1.67	3.00	7.00	267.00
Avg JI	0.00	0.38	0.55	0.71	1.00	0.01	0.43	0.65	0.84	1.00

Table 3: Comparison of average features per session before and after segmentation

6.1 Experiments over SQLShare

We implemented a first heuristic based in these 5 similarity indexes, tuned with simple thresholds whose setting is discussed in next subsection, taking the decision of segmenting or not using majority vote. This simple heuristic allowed to split the initial 451 sessions in 2,960 segments (called explorations). In the absence of ground-truth, we present in Table 3 a comparison of session features before and after session segmentation. A first remark concerns session length: extremely large sessions (maximum of 936) were split (new maximum is 78 queries). Indeed, more than half of the sessions were not fragmented and at 3rd quartile 1 session was split in 4 explorations. Some long and anarchic sessions (such as the one counting 936 queries) were split in a multitude of explorations. We can also highlight an increasing in the average number of common query fragments (Avg NCF) per session, as well as the average number of common projections, selections, aggregations and tables. This increasing is quite regular and visible for all quartiles. Relative edit distance (RED) and Jaccard Index (JI) also improved, as expected.

6.2 Experiments with ground truth

For this series of tests, we used workload datasets with ground truth. We expect our approach to find a segmentation that is close to the ground truth. As one of the datasets also contains timestamps, we compare to the heuristic used in the literature, i.e., the 30-minutes cutoff.

Datasets. We used three real workloads. Two of them are logs of multidimensional queries that we have used in our previous works [3], the third one is a log SQL queries used for clustering queries in [11]. We briefly present the datasets and we refer the readers to corresponding articles for more precisions.

The first dataset, named *Open* in what follows, consists of navigation traces collected in the context of a French project on energy vulnerability. These traces were produced by 8 volunteer students of a Master's degree in Business Intelligence, answering fuzzy information needs defined by their lecturer, to develop explorative OLAP navigations using Saiku[2] over three cubes instances. The main cube is organized as a star schema with 19 dimensions, 68 (non-top) levels, 24 measures, and contains 37,149 facts recorded in the fact table. The other cubes are organized in a similar way. From this experiment, we reuse 16 sessions, representing 28 explorations and 941 queries. The ground truth is a manual segmentation made by the lecturer based on some guessed notion of user goal and supported by timestamps. Notably, automatic segmentation was not the purpose of the work at the time manual segmentation was done.

The second dataset, named *Enterprise*, consists of navigation traces of 14 volunteers among SAP employees, in the context of a previous study on discovering user intents [5]. We set 10 business needs, and volunteers were asked to analyze some of the 7 available data sources to answer each of the 10 business needs, using a SAP prototype that supports keyword-based BI queries[3]. In total, this dataset contains 24 sessions, corresponding to 104 user explorations and accounting for 525 queries. Volunteers were explicitly requested to express to what information needs they were answering, which constitutes our ground truth for this dataset.

For these two datasets, the feature extraction was made as in [3], enriched with detection of the features pertaining to tables (NoT and NCT).

The third dataset, named *Exam*, consists of queries gathered over 2 years from student exams at University of Buffalo. Queries correspond to student's answers to 2 exam questions (one per year). A parser that allow to extract several query parts (including the ones we need) is also available for download. The parser also filters the queries that obtained low grades, proposing a total of 102 queries.

Notably, in Open and Enterprise datasets, users did not have to write any SQL code, contrarily to SQLShare or Exam. Indeed, Saiku and the SAP prototype generated queries from users high-level operations. However, in both cases, users devised real explorations. Conversely, queries in the Exam dataset are hand-written, but each student devised only one query. In order to simulate explorations, we made the hypothesis that the answers to a same exam question should mimic students attempts to find the correct answer. To this end, we arranged together the queries that answer a same exam question, obtaining 2 explorations (the ground truth for this dataset).

Dataset analysis and feature extraction. Table 4 compares the three datasets in terms of number of sessions, number of explorations (the ground truth), number of queries and summarizes features extraction. A first remark concerns the length of sessions. The Open dataset contains long sessions concerning few explorations while the Enterprise dataset contains shorter sessions concerning more explorations. Sessions length is actually dependent on GUI; while third party OLAP tools, like Saiku, log a new query for each user action (including intermediate drag-and-drops), the SAP prototype only logs final queries. In the Exam dataset, sessions and explorations were simulated. Regarding features, queries in the three datasets concern a quite small number of projections, selections, aggregations and tables. Relative edit

[2]http://meteorite.bi/products/saiku

[3]Patent Reference: 14/856,984 : BI Query and Answering using full text search and keyword semantics

	Open	**Enterprise**	**Exam**
Nb of sessions	16	24	1
Nb of explorations	28	104	2
Nb of queries	941	525	102
Avg queries per session	58	21	102
Avg queries per explor.	34	5	51
Avg explor. per session	2	4	2
Avg and range of NoP	3.62 [1,7]	2.18 [0,6]	1 [0,3]
Avg and range of NoS	1.33 [0,21]	0.76 [0,3]	1,57 [0,4]
Avg and range of NoA	1.34 [1,4]	1.14 [0,5]	0,77 [0,2]
Avg and range of NoT	3.28 [1,7]	2.03 [1,4]	3,02 [1,6]
Avg and range of NCP	3.16 [0,7]	1.34 [0,4]	0,22 [0,1]
Avg and range of NCS	1.13 [0,21]	0.46 [0,3]	0,07 [0,1]
Avg and range of NCA	1.17 [0,4]	0.77 [0,3]	0,09 [0,1]
Avg and range of NCT	2.97 [0,7]	1.46 [0,4]	2,57 [0,6]
Avg and range of RED	3.85 [0,19]	2.09 [0,25]	6.78 [1,11]
Avg and range of JI	0.57 [0,1]	0.79 [0,1]	0.31 [0,0.86]

Table 4: Characteristics of the 3 datasets

distance (RED) and Jaccard index (JI) illustrate that queries are more similar in the Enterprise dataset and highly dissimilar in the Exam dataset. The latter observation actually contradicts our hypothesis on the Exam dataset (stating that students' answers to a same exam question should be similar).

Threshold setting. As for the SQLShare dataset, we computed the 5 similarity metrics and used them (with voting strategy) for segmenting sessions. We tested different thresholds for voting. We proceeded as follows: we calculated the distribution of values for each metric and we used as threshold the value at k-percentile, with k varying in {0, 5, 10, 15, 20, 25, 30}.

The thresholds that provided better results were those at 0-, 15- and 5-percentile for the Open, Enterprise and Exam datasets respectively. These thresholds reflect the relationship between number of explorations to find and number of queries, as well as the similarity among consecutive queries. Indeed, the Open dataset contains many queries and few explorations (i.e., a few segments to find); small thresholds are best adapted. Conversely, the Enterprise dataset needs to be more segmented as the average number of queries per exploration is low; higher thresholds do better. With the same reasoning, 0-percentile should provide convenient thresholds for the Exam dataset. However, as queries inside an exploration are dissimilar, the only segmentation to find is not detected first (many intra-exploration breaks are proposed before); 5-percentile provides better results.

We remark that more precise thresholds could be learned with supervised machine learning techniques (e.g. classification). We intentionally avoid this computation because in real applications (as SQLShare) we do not have a ground truth for learning and, when one exists, we risk over-fitting. An expert providing the ratio of queries per exploration (either intuitively or via preliminary tests) is more realistic. For the SQLShare dataset, we rely on statistics, like number of queries per session (see Table 2) and the dataset description in [9] for setting the threshold as values at 30-percentile.

In the remaining tests, we use values at 0-, 15- and 5-percentile as thresholds for the Open, Enterprise and Exam datasets, respectively.

	Open	**Enterprise**	**Exams**	**Open** (timestamp)
Accuracy	0.98	0.88	0.94	0.97
Precision	1	0.78	0.17	1
Recall	0.42	0.63	1	0.25
F-measure	0.42	0.48	0.17	0.25
ARI	0.75	0.77	0.54	0.75

Table 5: Segmentation results for our approach on the 3 datasets and the timestamp-based approach (rightmost column)

Segmentation quality. For each dataset, we compared the obtained segmentation to the ground truth, measuring segmentation quality in two complementary ways. The first one, more demanding, compares the position of session breaks, indicating if both the segmentation and the ground truth coincide in the very same positions. To this end, we build binary vectors (a position corresponding to a query in the session), where a 1 means that a new segment starts at that position. We do the same for the ground truth and we compared both vectors in terms of accuracy, precision, recall and f-measure. The second way is the Adjusted Rand Index (ARI), a popular measure of clustering quality. It is based on the comparison of pairs of queries in the session, observing whether they belong to the same fragment and to the same exploration in the ground truth.

We report the results in Table 5. As expected, results are very good in terms of accuracy, mainly explained because classes are unbalanced (the number of *no-break* is higher than the number of *break*. Relative low values for F-measure while high values of ARI indicate that the exact break is not always found, while the overall segmentation reminds good. The bad results for the Exams dataset are mainly explained by the dissimilarities among queries of a same exploration (i.e. answers to a same exam question). The conclusions about query similarity presented in [11] confirm this fact.

Comparison to timestamp-based approach. In order to compare our approach to the one used in the literature, we implemented a second heuristic that segments sessions when there is a 30-minutes delay between queries. The Open dataset, the only one containing timestamps, was used for this test. Results are reported in Table 5, the right-most column corresponding to the timestamp-based approach. They are comparable in terms of accuracy and ARI but much lower in terms of f-measure. Note that 1 for precision means that all breaks found are also breaks in the ground truth. In other words, there are no big delays inside explorations, which makes sense. However, the timestamp-based approach fails to detect 75% of the breaks (when the user changes its topic of study in a briefer delay).

Analysis of similarity metrics. Finally, we investigated the quality of the 5 proposed similarity metrics, by studying the correlation of their vote (when metric value is lower than the corresponding threshold) with respect to the breaks in the ground truth. Results are presented in Table 6.

Jaccard and CF indexes are the more correlated in the Enterprise dataset. Both of them are highly correlated in the Open dataset, CT index being even better. Edit and Cosinus indexes are less correlated in one of the datasets. As expected, correlation of all measures is low in the Exam dataset, where segmentation is of bad quality. Interestingly, the most influencing metrics, the

	Open	Enterprise	Exams
Edit index	0.34	0.62	0.05
Jackard index	0.86	0.73	0.04
Cos index	0.75	0.32	0.13
CF index	0.86	0.69	0.10
CT index	0.90	0.50	0.01

Table 6: Correlation between similarity metrics and ground truth for the three datasets.

	Open	Enterprise	Exams
Edit index	0.23	0.69	0.16
Jackard index	1.00	0.99	0.66
Cos index	0.87	0.49	0.36
CF index	1.00	0.89	0.91
CT index	0.95	0.65	0.74

Table 7: Correlation between similarity metrics and final vote for the three datasets.

ones more correlated with the final vote (as shown in Table 7) are also Jaccard, CF and CT indexes.

7 CONCLUSION

This paper discussed the problem of fragmenting sequences of SQL queries into meaningful explorations when only the query text is available, and it is not possible to rely on query timestamps.

We characterized queries as a set of simple features and defined five similarity indexes with respect to previous queries in the session. A simple heuristic, based on the similarity indexes, allowed to split long and heterogeneous sessions into smaller explorations where queries have more connections.

Even if our preliminary results are promising, further aspects should be investigated:

- Study further query features. We would like to test the common fragments of a query w.r.t. close queries in the session (not only the previous one). Comparing query results seems important, even if it is not possible for many queries (because the database instance is not part of the SQLShare workload).
- Study other similarity indexes and tune thresholds.
- Discard preliminary hypothesis about chronological ordering of queries and deal with query similarity beyond it.
- Test other ways of fragmenting, in particular via clustering methods.

Our long term goal is to measure the quality of SQL explorations, allowing the detection of focus/exploratory zones, the discovery of latent user intents, the recommendation of next queries, among other applications.

REFERENCES

[1] B. D. Bhattarai, M. Wong, and R. Singh. Discovering user information goals with semantic website media modeling. In *MMM (1)*, volume 4351 of *Lecture Notes in Computer Science*, pages 364–375. Springer, 2007.

[2] S. Chaudhuri and V. R. Narasayya. Self-tuning database systems: A decade of progress. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 3–14, 2007.

[3] M. Djedaini, K. Drushku, N. Labroche, P. Marcel, V. Peralta, and W. Verdeau. Automatic assessment of interactive OLAP explorations. *To appear in Information Systems*, 2019. https://doi.org/10.1016/j.is.2018.06.008.

[4] M. Djedaini, N. Labroche, P. Marcel, and V. Peralta. Detecting user focus in OLAP analyses. In *Advances in Databases and Information Systems - 21st European Conference, ADBIS 2017, Nicosia, Cyprus, September 24-27, 2017, Proceedings*, pages 105–119, 2017.

[5] K. Drushku, N. Labroche, P. Marcel, and V. Peralta. Interest-based recommendations for business intelligence users. *To appear in Information Systems*, 2019. https://doi.org/10.1016/j.is.2018.08.004.

[6] M. Eirinaki, S. Abraham, N. Polyzotis, and N. Shaikh. Querie: Collaborative database exploration. *IEEE Trans. Knowl. Data Eng.*, 26(7):1778–1790, 2014.

[7] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015.

[8] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of data exploration techniques. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 277–281, 2015.

[9] S. Jain, D. Moritz, D. Halperin, B. Howe, and E. Lazowska. Sqlshare: Results from a multi-year sql-as-a-service experiment. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 281–293, 2016.

[10] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: Context-aware autocompletion for SQL. *PVLDB*, 4(1):22–33, 2010.

[11] G. Kul, D. T. A. Luong, T. Xie, V. Chandola, O. Kennedy, and S. J. Upadhyaya. Similarity metrics for SQL query clustering. *IEEE Trans. Knowl. Data Eng.*, 30(12):2408–2420, 2018.

[12] H. V. Nguyen, K. Böhm, F. Becker, B. Goldman, G. Hinkel, and E. Müller. Identifying user interests within the data space - a case study with skyserver. In *EDBT*, pages 641–652. OpenProceedings.org, 2015.

[13] O. Romero, P. Marcel, A. Abelló, V. Peralta, and L. Bellatreche. Describing analytical sessions using a multidimensional algebra. In *Data Warehousing and Knowledge Discovery - 13th International Conference, DaWaK 2011, Toulouse, France, August 29-September 2,2011. Proceedings*, pages 224–239, 2011.

[14] V. Singh, J. Gray, A. Thakar, A. S. Szalay, J. Raddick, B. Boroski, S. Lebedeva, and B. Yanny. Skyserver traffic report - the first five years. Technical report, December 2006.

[15] H. van den Brink, R. van der Leek, and J. Visser. Quality assessment for embedded SQL. In *SCAM*, pages 163–170. IEEE Computer Society, 2007.

[16] A. Vashistha and S. Jain. Measuring query complexity in sqlshare workload. https://uwescience.github.io/sqlshare/pdfs/Jain-Vashistha.pdf.

[17] R. W. White. *Interactions with Search Systems.* Cambridge University Press, 2016.

[18] M. Wong, B. Bhattarai, and R. Singh. Characterization and analysis of usage patterns in large multimedia websites. Technical report, 2006.