Benchmarking Cloud-based Tagging Services

Tanu Malik, Kyle Chard, Ian Foster

Computation Institute,
University of Chicago and Argonne National Laboratory
5735 S Ellis. Ave, Chicago, IL 60637
tanum@ci.uchicago.edu, kyle@ci.uchicago.edu
foster@anl.gov

***Abstract*—Tagging services have emerged as a useful and popular way to organize data resources. Despite popular interest, an efficient implementation of tagging services is a challenge since highly dynamic schemas and sparse, heterogeneous attributes must be supported within a shared, openly writable database. NoSQL databases support dynamic schemas and sparse data but lack efficient native support for joins that are inherent to query and search functionality in tagging services. Relational databases provide sufficient support for joins, but offer a multitude of options to manifest dynamic schemas and tune sparse data models, making evaluation of a tagging service time consuming and painful. In this case-study paper, we describe a benchmark for tagging services, and propose benchmarking modules that can be used to evaluate the suitability of a database for workloads generated from tagging services. We have incorporated our modules as part of OLTP-Bench, a cloud-based benchmarking infrastructure, to understand performance characteristics of tagging systems on several relational DBMSs and cloud-based database-as-a-service (DBaaS) offerings.**

I. Introduction

Information is created, shared and organized by humans in fundamentally different ways. Systems, however, store data, corresponding to information, in strict, pre-defined ways as governed by the underlying data model. This difference in logical presentation of information versus physical storage of data presents a discord in information organization, especially for data resources on the Web, affecting the ability to search, use, and share data and information with other users.

Tagging, whereby users add meaningful information to data resources, presents a natural and intuitive way to resolve this discord. Its usefulness has been demonstrated by several web sites, such as Delicious [1], Flickr [2], Technocrati [3], that have forsaken list-oriented and hierarchical views of Web resources to a more natural presentation created by using index terms and tags, often provided by users. Resources such as bookmarks, photos, and blogs are now organized and shared in ways that are intuitive to users thus increasing the value of their content. Tagging is also gaining wide attention in science communities as means to organize large volumes of ad hoc datasets resulting from an iterative process of data access, transformation, and integration.

To support this vital need, several cloud-based tagging services have emerged, such as FluidInfo [4], Magneto [5], Globus Catalog [6], and Zenodo [7] that enable users to describe and share data resources in a variety of ways and in various contexts. Internally, these platforms use an entity-attribute-value store [8] that is hosted on the cloud to provide a scalable service. The entity-attribute-value store associates any number of tags (attributes) with a resource (entity), often leading to dynamic, sparse data that lacks a highly-defined top-down structure.

This entity-attribute-value store can be realized on a variety of cloud-based platforms by choosing either traditional relational databases or scalable NoSQL databases. There is no clear consensus, however, as to which cloud-based platform provides the most efficient support for dynamic, sparse data and thus can help build the most cost-effective tagging service. In this paper, we describe a tagging benchmark and develop an infrastructure that uses this benchmark to compare various cloud-based database offerings. Our objective is to determine the most suitable database technology for running a large-scale, high-performance, cost-effective cloud-based tagging service. We choose cloud-based database offerings because they not only provide a common platform but are frequently used to implement such services.

Benchmarking relational DBMSs and cloud-based database-as-a-service (DBaaS) offerings is currently being investigated by the OLTP-Bench project [9]. This project is creating an infrastructure for evaluating different databases against a variety of workloads by controlling their transaction rate, mixture, and skew dynamically during the execution of an experiment. The infrastructure currently includes ten workloads that are derived from synthetic micro benchmarks, popular benchmarks, and real world applications. All the workloads, however, consider a fixed database schema over the resources instead of an dynamic schema as in the case of a tagging workload. Thus, we extend the OLTP-Bench infrastructure to support a tagging workload. In particular, we add a framework for making schema choices and for rewriting queries based on the schema, and generate the data and query workload that OLTP-Bench needs to test a tagging workload against various cloud-based offerings.

The contributions of this paper are as follows:

- A data and query benchmark for cloud-based tagging services. This benchmark generates attributes and values for resources with varying distributions and thus allows for sparseness of varying degrees. The query model allows for creation and browsing of tags. It models a

978-1-4799-3481-2/14/$31.00 © 2014 IEEE

faceted browsing session in which query formulation and expansion are simulated through a Markovian process.
- A framework that measures sparseness of a tagging dataset, and based on sparseness materializes several potential database schemas and queries suitable for different kinds of relational database.
- A thorough experimental evaluation of which database offering provides the best support for tagging services. This evaluation is achieved by plugging the schemas and queries in OLTP-Bench.

The remainder of the paper is structured as follows: Section II provides an overview of related work in this area; Section III-A describes tagging systems conceptually; Section III-B describes the benchmark workload generation process, and Section III-C describes the framework for creating multiple schemas. Section IV-A describes experimental setup with brief overview of the OLTP-Bench project and the modifications we made to it to support a tagging workload. We present our experimental results, comparing performance in transactions per second and throughput across databases in Section IV-B. We conclude in Section V.

II. Related Work

We focus here on tagging services available with websites, such as YouTube, Flickr, Globus, Zenodo, etc., that categorize, browse, and discover information about resources. These services are different from Twitter tags in which resources/messages are created to correspond to trending hashtags [16]. In Twitter, messages are tagged resulting in transient, trending social groups. In organizational tagging, user-described, semantically-meaningful keywords are attached to resources that result, over time, in a folksonomy over resources. We focus on organizational tagging, including user-defined, system-generated, and content-related tags. We also incorporate values with tags that are essential to convey precise information about the tag. For instance, longitude and latitude tags embedded in image files are incomplete without the actual values.

Our *resource, tag, value* data model corresponds closely to the *entity, attribute, value* (EAV) data model in which a few attributes, amongst the numerous available attributes, are chosen for describing entities. The difference in taxonomy arises due to prevalence of the former model in applications and the latter model in storage systems. The Resource Description Framework (RDF) model (*subject, predicate, object*) generalizes the EAV model in that in RDF objects are also classes similar to entities/subjects, and not scalar values as in EAV model [10]. The storage and schema design issues that we consider in this paper also apply to RDF data, and have been investigated in [11], [12], and [13], demonstrating significant performance improvements by choosing a given schema configuration over another for a given set of information-retrieval queries. However, current RDF benchmarks do not provide mechanisms to evaluate schema-design issues, but choose a fixed three-triple schema.

The cloud is becoming a *de facto* platform for hosting data services, including tagging services. Using the cloud to host a service requires an understanding of performance of its pay-per-use cost models, shared multi-tenant infrastructures, and system configuration, many of which are beyond user control. Benchmarking serves as the only option to determine the performance of the cloud [9]. Many benchmarks have been developed for the cloud.

The CloudStone benchmark considers a Web 2.0 social application (*Olio*) in which resources are events and users create tags on events and add comments and ratings as tags [14]. CloudStone, however, benchmarks the web server and considers database tuning as a complex process (described as out of scope for the paper) thus scoping out the impact of database schemas and corresponding workloads. Database-specific benchmarks, such as [15] have primarily focused on TPC-C benchmarks and distributed settings. The OLTP-Bench is a relatively new effort that is creating a necessary infrastructure for testing a variety of relational workloads and monitoring performance and resource consumption of a clod-based database under test. However, it does not support tagging like workloads[1].

III. The Tagging Benchmark

The study of tagging systems as described in [16] and [17] lays the foundations for our tagging service. In these works, the tagging model represents tags as keywords with no values. However, dedicated tagging services, such as FluidInfo and Magneto assume the more general tag-value model. Thus we describe a tagging model with values. To construct a benchmark, tag usage studies are needed. [16] conducts an analysis of tag usage over resources and by users in the context of Flickr. This analysis forms valuable descriptive groundwork, and we use this analysis to approximate usage distributions by concrete functions that naturally reflect the scenarios, e.g. logistics curves for modeling limited growth or power equations for power law distributions. All approximations have been done with the ZunZun data modeling tool and the Gnuplot curve fitting module.

A. The Tagging Model

Figure 1 shows the conceptual model for tagging systems. In this model, users or system agents assign tag-value pairs to a specific resource; tags and their values are represented as typed edges connecting users and resources. Resources may be also be connected to each other (e.g., as links between web pages) and users may be associated by a social network, or a set of affiliations.

Resources are identifiable objects that can be digitally or virtually represented, and in which tag-value pairs improve the semantic descriptions of the resource. We assume that resources are either supplied by the users of the system or provided internally by the system, as in the case of some workflow systems. Resources in the system can be linked to each

[1]The OLTP-Bench Twitter workload focuses on the graph structure of the data and not the tagging service

other independent of user tags. For example, if the resources represent web pages then they are connected by direct links or by common entities. Similarly if resources represent files then they may be connected, under a containment relationship, with a virtual directory resource.

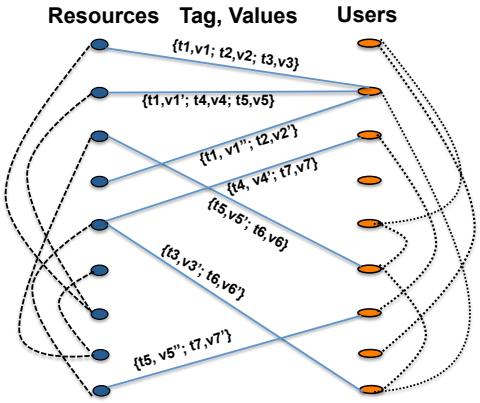


Fig. 1. The Tagging Model

Tags and their values are associated with resources in three explicit modes: (i) blind, (ii) viewable, or (iii) suggestive. In the blind mode, users/agents do not view/know tags assigned to the resource by other agents; In the view mode users/agents can see the tags that are already associated with a resource. In the suggestive mode, the tagging service suggests possible tags to users/agents. The suggested tags may be based on existing tags by the same user, tags assigned to the same resource by other users or based on an externally supplied vocabulary. Blind tagging may result in multiple, duplicate tags for the same resource from different users, often termed as the bag mode. Alternatively, viewable and suggestive tagging prevents repetitions, termed as the set mode.

Many tagging services establish access control over resources, tags, and their values. Permissions are typically expressed in the same way as Unix file systems (read, write), where permissions such as create and delete are included in the write permission. Policies associated with these permissions are generally defined at the individual or group level. Permissions may be inherited from other objects in the system, for instance a value may inherit the permissions of a tag or a resource, and a tag may inherit the permission of a resource. A tagging system may therefore enable permissions to be used in several different ways, for instance users may allow tagging of their resources or use of their tags by any user or they may place restrictions on tagging, for example allowing only self-tagging (i.e. users can only tag the resources they created) or even creating private values when resources and tags are accessible to others. In addition, the service may support policies on creation or removal of a resource (no one, anyone, the resource owner), a tag (no one, anyone, the tag owner or the resource owner), or a tag value (no one, anyone, the value owner, the tag owner, or the resource owner)

Creating a tag-value on a resource assumes that the corresponding tag exists in the system. Defining tags can be achieved by describing few important properties of tags: (i) their data type, (ii) whether they are single valued or multi-valued (arrays and lists are considered multi-valued). Tag values may be unique across all instances of that tag or non-unique.

Based on existing user studies and empirical results on Web-based tagging systems, such as Delicious, Technocrati, and Flickr, and our own experiences with creating tagging systems we see the following characteristics to be common across tagging systems:

- Some resources are more frequently tagged than others. Empirical results show that a power law holds for this phenomenon.
- Number of distinct tags owned by a user is relatively few. [17] showed that the probability that a Flickr user has more than 750 distinct tags is roughly 0.1%.
- Distinct tag usage increases with the increase in the number of users and resources in the system, with higher correlation coefficient with the number of resources than users.

B. Workload Generation

Unlike other conventional benchmarks, the tagging benchmark has no separate data and query generation phases; database tables and their attributes, i.e., tags, are decided by the incoming user workload leading to an openly-writable metadata storage system. We have implemented our synthetic workload generator program in Python. It takes into account all relationships and characteristics between resources, users, and tags that have been described in the previous section. We assume the most general options for the workload generator, in particular, no access control policy on resources, a suggestive mode of tagging and tag creation as optional to resource creation. Based on these options, the following user operations are supported as part of the workload generation phase:

- S1: Add a user.
- S2: A user uploads a resource.
- S3: A user creates a tag definition and uses this definition to tag a resource with a value.
- S4: A user queries for a resource.

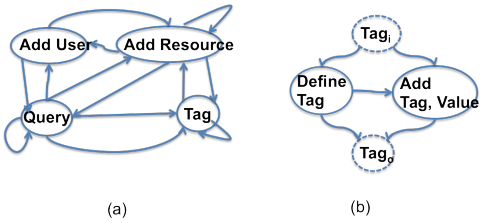


Fig. 2. The Markov Chain (a) States and some state transitions, (b) The composite tagging state.

Figure 2(a) shows the session-based, closed loop Markov-chain behind the synthetic workload generator. Each operation corresponds to a state, with the tagging state defined as a composite state consisting of sub-states: (i) creation of a tag definition, which creates a new tag, and (ii) tagging a resource

with a newly created or previously existing tag definition (Figure 2(b)). Similarly, the query state may consist of several kinds of queries in which the user issues several queries to find the most relevant resources. We describe different kinds of queries shortly. An $N \times N$ matrix defines the probability of transitions between the N states. In general, this matrix can be derived from the workload of a tagging service. For lack of a suitable workload, we initialize this matrix with different parametric values that represent a broad class of tagging services.

The probabilities in the transition matrix of the Markov chain are chosen so as to follow the three fundamental characteristics of tagging systems described in Section III-A. When a user queries for a resource to be tagged, the chosen resource is dictated by a power law. The transition probabilities of using an existing tag is always kept higher than the probability of creating a new tag definition. This bounds the number of distinct tags owned by a user for the session. Finally, the third characteristic is held by maintaining the internal transition probabilities in the tagging state as a function of the previous state, i.e., if tagging is followed by a resource upload then the transition probabilities are different than if tagging is followed by add user or query states.

In tagging services, querying for a resource may not be restricted to a single query but may involve a series of queries, each targeted towards satisfying users' goals towards retrieving the relevant resource. We do not assume any access control policy to be in place while searching for resources, so, given a query, search results include all resources that qualify. We enumerate various search query categories that a user may issue to obtain desired information. These queries correspond to the fourth user operation (S4):

- Search for resources with a given/multiple set(s) of tag 'T' and value(s) 'V'. This query can have multiple variations where both tag and value are specified through a variety of binary, logical and comparative operators.
- Find all other tags or popular tag-value pairs on a given resource.
- Suggest tags for a selected resource.
- Find resources that are "linked" to set of resources that have a tag 'T' and a value 'V'. This is a referential query where resources from the 'link' tag have to be joined with resources found from tag-value pair (T, V).
- Find resources where value like 'V' exists. This query fetches all tags with which the user is associated.

In a given query sequence, the likelihood of one query increases over the other based on the previous query issued. However, for lack of any suitable workload to determine these probabilities, we assume each query to be equally likely. We assume the user iterated over a length of sequence determined by a normal distribution.

C. A Framework for Evaluating Tagging Datasets

As is evident from Figure 2(b) tagging may modify the underlying database schema, adding attributes or inserting values. These modifications change the nature of the dataset: as new tags are defined, it increases the sparseness of the dataset, since most resources have non-null values for only a small number of the defined attributes. As existing tags are reused and new values are added, it decreases the sparseness of the dataset. This dynamic change in sparseness of the tagging dataset affects the efficiency of the tagging service and is a function of the choice of underlying database system.

If the underlying database is a NoSQL system, then it is challenging to benchmark, since prominent NoSQL databases do not provide built-in support for joins, which are needed to answer the tagging query workload (joins are needed to search for similar tags on a given resource or to recommend other resources to users). If NoSQL databases are used for tagging services, then joins must be supported in the application logic, subjecting them to several user code and programming language optimizations. This makes it challenging to benchmark such systems in a uniform way. Relational database provide native and efficient support for joins and are also favored by most commercial tagging services, such as FluidInfo, Globus Catalog, Magneto and Zenodo. Therefore we focus on relational systems as a database for tagging services, and describe how to benchmark them for sparse, dynamic schemas and tagging queries involving joins.

Several approaches can be taken to support sparse tagging datasets in relational databases. The simplest approach is to store them in a horizontal schema, where in each column represents a distinct attribute and each row represents an object. Some of the values are null. Though simple, if the underlying database does not support nulls efficiently, this approach may waste a lot of space and makes query scans highly inefficient. An alternative is to use vertical schema, which eliminates null values. However, this approach often leads to complex queries and poor query performance. Yet another approach is to group commonly occurring tags together, and relegate the rest of attributes to a single sparse horizontal table or store them as vertical tables. An orthogonal approach is to consider resource, tag and value as a triple and store all such triples in a single table. This approach leads to several self-joins on the table, when answering tagging queries. The essential trade-off between these approaches is the ability to store null values versus the ability to conduct join queries. Different databases show differing performance characteristics for this tradeoff and therefore it is useful to determine what storage model/schema to choose in a given database system.

The benchmark framework, given a tagging (query and tag) workload, generates a variety of schemas for a given database system. In general all schemas can be generated; to avoid generation of schemas which will result in poor performance, the framework follows the following rules. First, it measures sparseness of the generated tagging dataset using a sparseness metric defined as:

$$s = \frac{\mid tag\_T\_is\_set\_with\_a\_value\_V \mid}{\mid tags \mid * \mid resources \mid} \quad (1)$$

The value of the metric, which ranges from [0,1], determines the database schema(s) that will be manifested in the rela-

tional database. When the sparseness metric is very low, the framework generates vertically partitioned and triple schema. When the metric is very high, it generates horizontal and triple schema. In between, the framework generates all four schemas: a vertically partitioned, horizontal, triple and attribute schema. The low, and high values are configurable and set to 0.2 and 0.8, respectively. The attribute schema is inherently a binary schema to allow tags such as longitude and latitude to be coalesced together. The framework currently combines any two attributes, which reduce the sparseness measure of the corresponding horizontal schema by half. Note in general, many attributes can be combined, but to keep complexity low, we generate binary tables.

A query module reads the database schema and translates the given queries into the specific schema and appropriate dialect of the chosen database. This rewriting is primarily required for the vertical and the attribute table approach, since queries that are simple over the horizontal schema become more complex in the vertical approach. Simple projection queries over a horizontal table are transformed into selection queries over a vertical table. The projection query for vertical schema retrieves, for each entity selected in the query, the rows corresponding to the non-null attributes specified in the query. A select query over a vertical schema is complex because it has to retrieve all of the attributes of an entity that match the selection predicate.

IV. Cloud-based Experiments

We present experiments for the tagging workload as conducted on several cloud-based offerings. Our primary objective through these experiments is to demonstrate that the framework for evaluating tagging datasets is a required component when choosing amongst many DBaaS offerings on the cloud. As with other benchmarks, our objective is not to compare or judge alternative DBMSs and DBaaS offerings.

A. Experimental Setup

Our experimental setup consists of the workload generator and the benchmark framework on a single machine. The generated schema and the workload is plugged into OLTP-Bench, which is deployed on Amazon's EC2 platform. We briefly describe the OLTP-Bench infrastructure and the EC2 setup.

OLTP Benchmarking Infrastructure The OLTP-Bench consists of a client-based infrastructure that executes web workloads on relational databases deployed on the cloud. The system consists of a configuration file that instructs the system on how to connect to the DBMS under test, what experiment to run, the level of parallelism desired, and how to vary rate and mixture over time. We plug the workload generator into the centralized Workload Manager for programmatic generation of the workload. The Manager generates a work queue, which is consumed by a user-specified number of threads to control parallelism and concurrency. All the threads are currently running on a single machine. At this point we have not plugged the OLTP-Bench into its SQL Dialect Management; this will be a focus of future work. Currently, we generate the workload a priori knowing the kind database for which the workload is targeted. The generated workload is then issued to the OLTP-Bench Workload Manager.

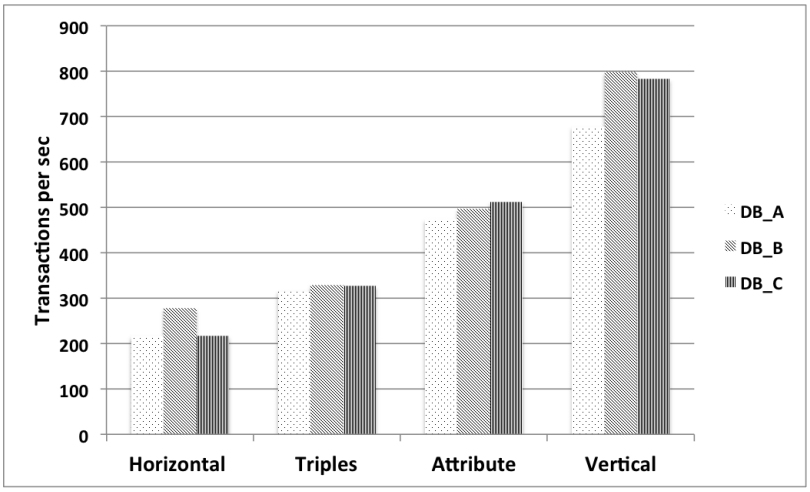
EC2 Setup The server DBMSs were deployed within the same geographical region, with a single instance dedicated to workers and collecting statistics and another instance running the DBMS server. For each DBMS server, we used a four virtual core instance with an 8GB buffer pool (this was sufficient to accommodate the working-set size of our workloads). We flushed each system's buffers before each experiment. All of the changes made by each transaction were logged and flushed to disk by the DBMS at commit time. As recommended by OLTP-Bench, to mitigate noise in cloud environments [18], we ran all of our experiments without restarting our EC2 instances (where possible), and executed the benchmarks multiple times and averaged the results.

Database Setup We experiment with three databases: MySQL with InnoDB version 5.6, Postgres version 9.2, and SQL Server 2010, hereby termed **DB_A**, **DB_B**, and **DB_C** in no particular order. The choice of these three databases is dominated by the fact that the first two are open-source and provide efficient space allocation for NULLs, and SQL Server, though commercial, offers the 'SPARSE' language construct for sparse datasets. We provisioned the databases on a medium instance of Amazon EC2. The memory and buffer configuration was kept the same across all databases. The schema specification was translated to the target database and indexes were included as part of all tables. In the horizontal table schema, an index was created on the primary key of resource ids. In the vertical schema, the tag tables are sorted based on the resource ids so that fast merge joins are maintained. There is a clustered B+ tree index on resource identifiers. Multi-valued attributes are represented through multiple rows in the table with the same subject and different object value. Attribute tables also have clustered indexes on primary resource ids.

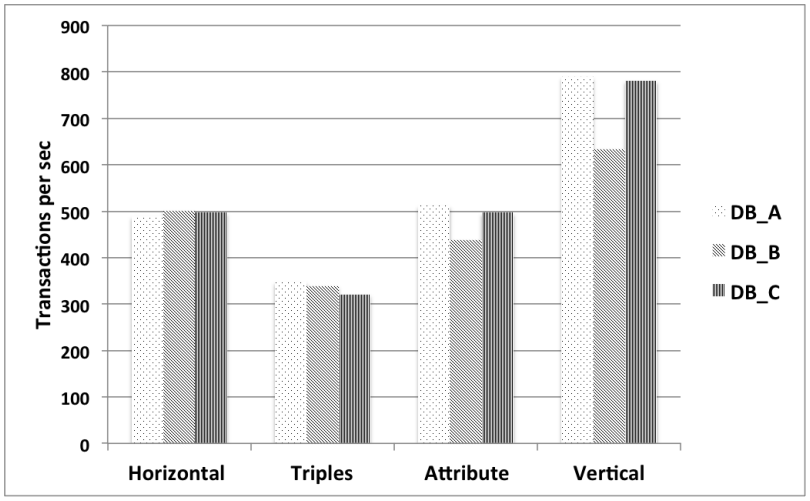
B. Results

We include three experimental results, each of which answers the following questions that arise when choosing database systems for tagging systems: (i) Given a schema and a tagging workload, which database offers the best performance (good overall throughput and low latency) and cost trade-off?, (ii) For a given schema, which database offers the best scalability as the data size increases?, and (iii) Given schema choices, do databases perform differently if the query workloads are rapidly changed?

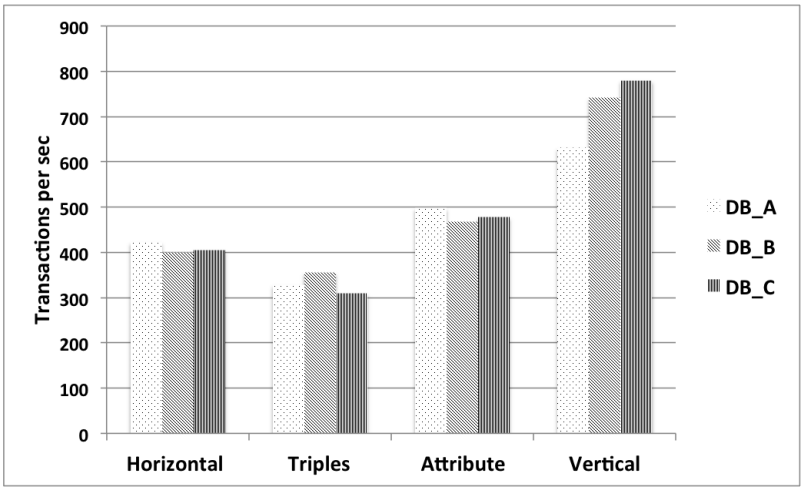
1) **Comparing Database SaaS Providers:** We answer this question in two parts: first, what is the appropriate database for a given schema, and second using a database schema for a given database what is its impact on performance and cost. To conduct this experiment, we varied the probabilities in the transition matrix for generating new tags or using existing tags, which resulted in datasets with differing values of the sparseness metric: 0.37, 0.55, and 0.77, respectively. Intuitively these



(a) For s = 0.37



(b) For s = 0.55



(c) For s = 0.67

Fig. 3. Comparing the Four Schemas Across Database Offerings

three values represent whether the user is creating new tags (lower value) or using existing tags and inserting new values for them (higher value). Since these values fall in the [0.2,0.8] range, all four schemas are potentially viable for satisfying the workload. The generated query workload is the same in all cases since query probabilities are kept the same. The workload generator was run for 30 minutes generating a total of 100k resources and 1000 tags.

Figure 3 (a), (b) and (c) show the results for different schemas. As shown by the figures, it is clear that both the vertical and attribute schema approach perform better (roughly by a factor of 1.5 and 2.5) than the triple schema approach in all databases. The horizontal schema performs worse when the sparseness metric is on the lower side (0.37). This is because only one of the query types in the query workload (the first query) leads to a simple selection in the triple schema. The rest lead to one or more self-joins, therefore decreasing its performance. The horizontal schema leads to a self-join query in one of the cases (the third query), but selection in all other cases. However, its performance decreases when there are many null values in the system. The primary contention is between vertical and attribute schema, and different databases perform better for both. Given vertical schema choice, **DB_B** performs better for low sparseness metric (0.37 and 0.55) but **DB_A** performs better for high sparseness metric. The reason for this change is that **DB_B** uses an index nested loops join to join keys while **DB_C** chooses to do a merge join. However, the index nested loop join is a poor choice when the sparseness metric is high. This, we believe, can be improved by keeping better statistics on the vertically partitioned table.

We next determine if the different schemas have an impact on performance vs cost tradeoff. We test and leverage OLTP-Bench to test against four different instance sizes on Amazon Relational Database Service (RDS) (See [9] for detailed instance description). We then ran each benchmark separately at its maximum speed for a total of 30 minutes. We show average maximum sustained throughput (transactions per second) and the ratio of transactions and cost. The experiment shows an interesting result that different schemas achieve peak performance for different instance sizes in **DB_A**. The reason is that different schemas pose different constraints on memory and CPU requirements for the same workload. Vertical and attribute schemas, inevitably require creation of temporary tables for storing intermediate results. Thus, they utilize a larger number of cores and more RAM with XL-HM, which is not the case for triple and horizontal schemas in which available resources remain un-utilized. Thus the latter can achieve peak performance with a large instance, while vertical and attribute schemas need XL-HM to achieve best performance.

2) **Scalability:** In this experiment, we turn our attention to the performance of the schemas in a given database as the size of data increases. For this experiment we ran the workload for an hour and generated a total of 10 million resources and 10 thousand tags. Figure 5 shows how the different schemes perform on **DB_B**. We see differing performance characteristics. For **DB_A** and **DB_B** vertical and attribute schemes scale linearly, while triple schema and horizontal schema scales sub-linearly. This is because all joins are linear for the vertically partitioned schemes (merge joins); triple and horizontal schema do not sort the intermediate results or most queries and therefore result in sub-linear performance. In **DB_C** all schemes perform super-linearly. This, we believe, is because of caching effects with the **DB_C** query optimizer.

3) **Changing Query Vs Tagging Workloads:** In this experiment, we determine how different cloud offerings compare in terms of changing workload composition. We change the workload by moving from a purely tag creation and tag generation workload to a purely query only workload, with no additional tags being generated. Again the four schemas are tested on the three database offerings. Here the result (Figure 6) is consistent across databases (so we report only one database result): the vertical and attribute schemas perform best for a mixed workload irrespective of the databases, while the triple schema performs best for a purely tagging workload. The horizontal schema works best when no additional tags are being added.

V. Conclusion

In this paper, we presented a benchmark for tagging services that have gained wide popularity in Web 2.0 applications as a means to present a semantic view of the underlying resources. Tagging services are also gaining wide attention in the scientific community as means to organize large volumes of ad hoc datasets resulting from iterative process of data access, transformation, and integration. In supporting Globus Catalog, a cloud-based tagging service, we have faced the issue of choosing the most cost-effective and efficient database offering on the cloud. The benchmark proposed in this paper has been used to test some of the cloud-based offerings for Globus Catalog. We have found the need for a benchmark framework that allows manifestation of different schemas as essential to evaluating the various databases.

Currently, we have not released source-code of the benchmark, but that is being undertaken as we report the results of the work. The benchmark, framework, and integrated OLTP-Bench infrastructure will soon be available from [19].

Acknowledgment

We would like to thank Carl Kesselman, Karl Czajkowski, and Rob Schuler at ISI for discussions on this subject. This work is supported in part by DOE through grant DE-AC02-06CH11357.

References

[1] (2013) Delicious. [Online]. Available: https://delicious.com/
[2] (2013) Flickr. [Online]. Available: http://www.flickr.com/
[3] (2013) Technocrati. [Online]. Available: http://technorati.com/
[4] (2013) Fluidinfo. [Online]. Available: http://fluidinfo.com/
[5] (2013) The Magneto Connect. [Online]. Available: http://www.magentocommerce.com/magento-connect/
[6] (2013) The Globus Catalog. [Online]. Available: http://catalog-alpha.globuscs.info
[7] (2013) Zenodo. [Online]. Available: http://zenodo.org/

[8] (2013) The EAV model. [Online]. Available: http://en.wikipedia.org/wiki/Entityattributevalue_model
[9] (2013) The OLTP-Bench. [Online]. Available: http://oltpbenchmark.com/
[10] (2013) The resource description framework. [Online]. Available: http://www.w3.org/RDF/
[11] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach, "Sw-store: a vertically partitioned dbms for semantic web data management," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 18, no. 2, pp. 385–406, 2009.
[12] K. Hose, R. Schenkel, M. Theobald, and G. Weikum, "Database foundations for scalable rdf processing," in *Proceedings of the 7th international conference on Reasoning web: semantic technologies for the web of data*. Springer-Verlag, 2011, pp. 202–249.
[13] J. J. Levandoski and M. F. Mokbel, "Rdf data-centric storage," in *Web Services, 2009. ICWS 2009. IEEE International Conference on*. IEEE, 2009, pp. 911–918.
[14] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, and D. Patterson, "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0," in *Proc. of CCA*, 2008.
[15] D. Kossmann, T. Kraska, and S. Loesing, "An evaluation of alternative architectures for transaction processing in the cloud," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 579–590.
[16] J. Huang, K. M. Thornton, and E. N. Efthimiadis, "Conversational tagging in twitter," in *Proceedings of the 21st ACM conference on Hypertext and hypermedia*. ACM, 2010, pp. 173–178.
[17] C. Marlow, M. Naaman, D. Boyd, and M. Davis, "Ht06, tagging paper, taxonomy, flickr, academic article, to read," in *Proceedings of the seventeenth conference on Hypertext and hypermedia*. ACM, 2006, pp. 31–40.
[18] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 460–471, 2010.
[19] (2013) The globus github. [Online]. Available: https://github.com/globusonline/

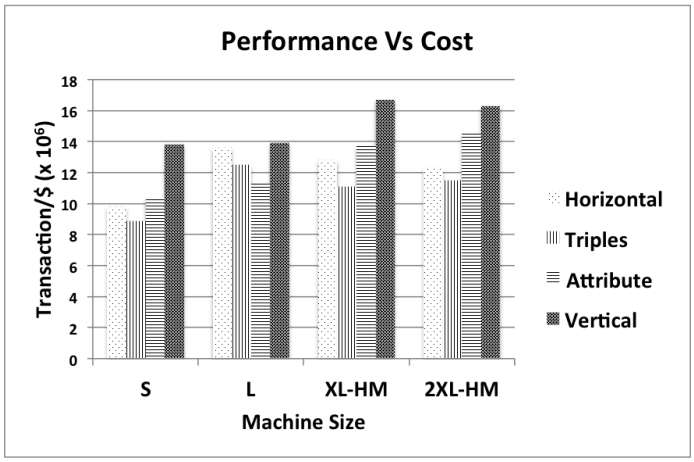


Fig. 4. Performance Vs Cost Tradeoff for **DB_A**

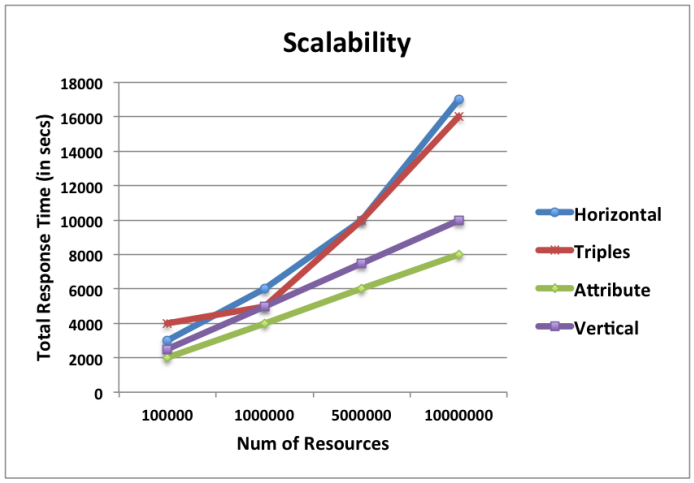


Fig. 5. Increasing Number of Resources for **DB_B**

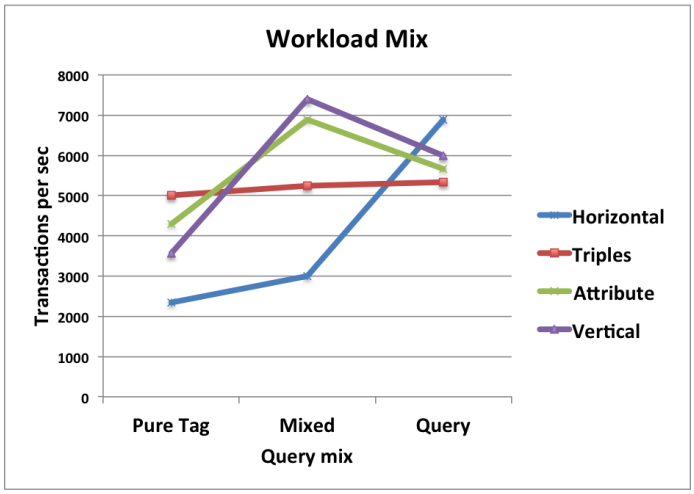


Fig. 6. Changing the Workload Mix for **DB_C**