J Supercomput (2013) 63:722–736
DOI 10.1007/s11227-011-0717-y

A workload-driven approach to database query processing in the cloud

Adnene Guabtni · Rajiv Ranjan · Fethi A. Rabhi

Published online: 24 November 2011

Abstract This paper is concerned with data provisioning services (information search, retrieval, storage, etc.) dealing with a large and heterogeneous information repository. Increasingly, this class of services is being hosted and delivered through Cloud infrastructures. Although such systems are becoming popular, existing resource management methods (e.g. load-balancing techniques) do not consider workload patterns nor do they perform well when subjected to non-uniformly distributed datasets. If these problems can be solved, this class of services can be made to operate in more a scalable, efficient, and reliable manner.

The main contribution of this paper is a approach that combines proprietary cloud-based load balancing techniques and density-based partitioning for efficient range query processing across relational database-as-a-service in cloud computing environments. The study is conducted over a real-world data provisioning service that manages a large historical news database from *Thomson Reuters*. The proposed approach has been implemented and tested as a multi-tier web application suite consisting of load-balancing, application, and database layers. We have validated our approach by conducting a set of rigorous performance evaluation experiments using the Amazon EC2 infrastructure. The results prove that augmenting a cloud-based load-balancing

A. Guabtni · F.A. Rabhi
School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia

A. Guabtni
e-mail: aguabtni@cse.unsw.edu.au

F.A. Rabhi
e-mail: fethir@cse.unsw.edu.au

R. Ranjan (✉)
Information Engineering Laboratory, CSIRO ICT Center, Building 108, Australian National University, Canberra, Australia
e-mail: rajiv.ranjan@csiro.au

service (e.g. Amazon Elastic Load Balancer) with workload characterization intelligence (density and distribution of data; composition of queries) offers significant benefits with regards to the overall system's performance (i.e. query latency and database service throughput).

Keywords Range query processing · Load balancing · Data density · Cloud computing

1 Introduction

Cloud computing is the latest evolution of computing, where IT capabilities are offered as services. Cloud computing [1–4] delivers infrastructure, platform, and software (application) as services, which are made available as subscription-based services in a pay-as-you-go model to consumers. These services in industry are respectively referred to as Infrastructure as a Service (IaaS) [5–7], Platform as a Service (PaaS) [2, 8–10], and Software as a Service (SaaS) [11]. A technical report [1] published by University of Berkeley in February 2009 states that "Cloud computing, the long-held dream of computing as a utility, has the potential to transform a large part of the IT industry, making software even more attractive as a service".

This paper is concerned with the problem of hosting data provisioning services over Cloud infrastructures. Most efforts so far take advantage of database replication and the reduced maintenance, flexible elasticity, and high availability of on-demand infrastructure [12] offered by Cloud providers such as Amazon EC2, Google AppEngine, Microsoft Azure, GoGrid, and others. Although deploying different components (database, web server, application server) of a data provisioning service on Cloud resources (compute server, storage) is trivial, deriving cost-effective and efficient performance (query processing time, utilization, and throughput) out of these deployments is challenging due to various issues.

Range queries [13] are extensively common workload type in context of data provisioning services. A range query over a large data set can be computationally expensive and if not properly handled may lead to poor performance (e.g. query processing time). The majority of Database Management Systems (DBMS) handle a range query by partitioning it across multiple replicas for reducing the query processing time. However, query processing has a cost related to the hardware and software investments required for maintaining large database replications. In addition to the massive workloads generated by range queries, scalability requirements [14] can change very quickly and may cause a degradation in performance depending on the type and/or the number of requests made by users. Cloud computing [12] offers the necessary flexibility and adaptability to handle such issues and many companies are using the cloud as a deployment environment for replicated databases to ensure the scalability of their data services [15]. Indeed, scalability of services based on replicated cloud databases is made possible using its load balancing and auto-scaling capabilities. However, auto scaling can have an substantial cost when a large number of database replicas are instantiated. Therefore, the key efficiency factor for data provisioning is to optimally balance the workload across all database replicas to enhance the performance and reduce costs.

The specific aim of this paper is to investigate how cloud computing could benefit the objective of providing efficient data provisioning services [16–19]. Two aspects of cloud computing and one aspect of range query processing are of particular interest to this paper. Firstly, cloud-based load balancers could be used to optimally balance the workload of processing range query over a given set of database replicas. Secondly, the auto-scaling capabilities (elasticity) of a cloud infrastructure could be used to adapt the capacity of a data provisioning service to address a variable query rate. Finally, range query partitioning is of particular interest to split a range query into smaller ones such that they can be distributed across the available database replicas in a load-balanced fashion.

The main contribution of this paper is a novel approach combining cloud-based load balancing techniques and density-based range query partitioning for efficient range processing over a cloud infrastructure with a case study of a large historical news database. The proposed approach has been implemented using a Service Oriented Architecture (SOA) over news databases replicated on the Amazon EC2 infrastructure. The rest of the paper is structured as follows. Section 2 first discusses the specific characteristics of the used data sets (historical news), i.e. its non-uniform data distribution. Section 3 presents the proposed approach for density based range query partitioning and distribution. Section 4 discusses experimental results. Section 5 discusses related work. Finally, Sect. 6 summarizes the results of this work and presents our conclusions.

2 Research problem analysis

As our proposal is based around range query processing, this section describes the characteristics of the data sets and queries used for this study. A data set, noted in this paper as *NDS*, contains textual news announcements that are captured over span of time. The following properties are useful for characterizing a news announcement n_i in a news dataset *NDS*:

- $t(n_i)$ is the publication date and time for n_i.
- $T(n_i)$ is the set of tags associated with n_i. Each news announcement n_i can be linked with one or more tags that helps in news indexing and categorization. These tags are defined by concerned news providers. This study particularly focuses on Reuters Instrument Codes, or RICs applied by Thomson Reuters for identifying financial instruments and indices. Examples of RICs are company codes such as AAPL.O (for Apple), MSFT.O (for Microsoft), and IBM.N (for IBM).
- $s(n_i)$ the size of storage required to store n_i in a relational database system.

We define $\prec$ as an absolute precedence operator such that for any two news announcements n_1 and n_2, if $t(n_1) < t(n_2)$, then $n_1 \prec n_2$ also holds true. Therefore, a news data set can be formalized as a sequence of news announcements $NDS = \{n_1 \prec n_2 \prec \cdots \prec n_k\}$, where n_1 is the first and n_k is the last news announcement, respectively. A typical search query over a news database requires a time interval (e.g. between 01/01/2003 and 31/12/2007) and a list of search tags. Such queries are better known as range queries. More precisely, a range query is a common database operation that retrieves all records where some value is between an

upper and a lower boundary (in our case, it is a date/time interval). As an example, one would request all news tagged with a particular company code (e.g. MSFT.O for Microsoft) between January 2003 and December 2007. In this case, the range concerns the publication (announcement) date of the news articles. We formalize a range query on a news dataset as follows.

Definition 1 (Range query) A range query q on a news dataset NDS searches for news announcements with tags in $TS(q)$ and that were published within the time interval $\{t_s,\ t_e\}$. The result set obtained by executing q is denoted as $RS(q)$:

$$RS(q) = \{\forall\, n \in NDS\ /\ t_s(q) \leq t(n) \leq t_e(q);\ \ T(n) \in TS(q)\}$$

Further, the measured processing time of a range query q is denoted as $pt(q)$, and the size of its result set is given by the following function:

$$Size(RS(q)) = \sum_{n \in RS(q)} s(n)$$

Range queries are used in various service domains such as Internet-based search engines, auctions, and gaming. Usually, users issuing queries in aforementioned services do not experience significant delays. That is due to the fact that, services do not immediately return all the results to the user as they apply complex pagination techniques in which a limited number of records are retrieved on each page. Following that, the results are sorted and only the N most recent records are shown to the user. These pagination techniques are however not adapted to our case in which users need to apply further processing to the entire result set (e.g. statistical analysis). In such a situation, processing range queries without pagination can be computationally expensive when applied to large data sets.

We define a query partitioning function $QSPL^k(q) = \{q_1, \ldots, q_k\}$ as the partitioning of a query q into k queries distributed on k database replicas. Existing query distribution mechanisms, built in Database Management Systems, assume a uniform data distribution and, therefore, range query partitioning based on equal ranges is defined as follows:

Definition 2 (Query Partitioning based on equality of ranges) The Query Partitioning Function based on equality of ranges for k database replicas, noted $QSPL^k_{eqr}() : q \longrightarrow \{q_1, \ldots, q_k\}$, ensures that:

$$\forall q = \{t_s,\ t_e, TS\}, \qquad \forall q_k = \{t_s^k,\ t_e^k, TS\} \in QSPL^k_{eqr}(q), \quad \left(t_e^k - t_s^k\right) \simeq (t_e - t_s)/k.$$

The majority of existing relational database systems (MySQL, PostgreSQL, Oracle, etc.) expose extensive data replication and query partitioning capabilities. However, these approaches assume uniform distribution of data and, therefore, are not capable of generating sensible partitioning and mapping of queries for non-uniform data.

As a case study, we have used the Thomson Reuters news database provided by an Australian-based company SIRCA [20] which provides high frequency financial market and news data in the form of eResearch services to over 200 universities,

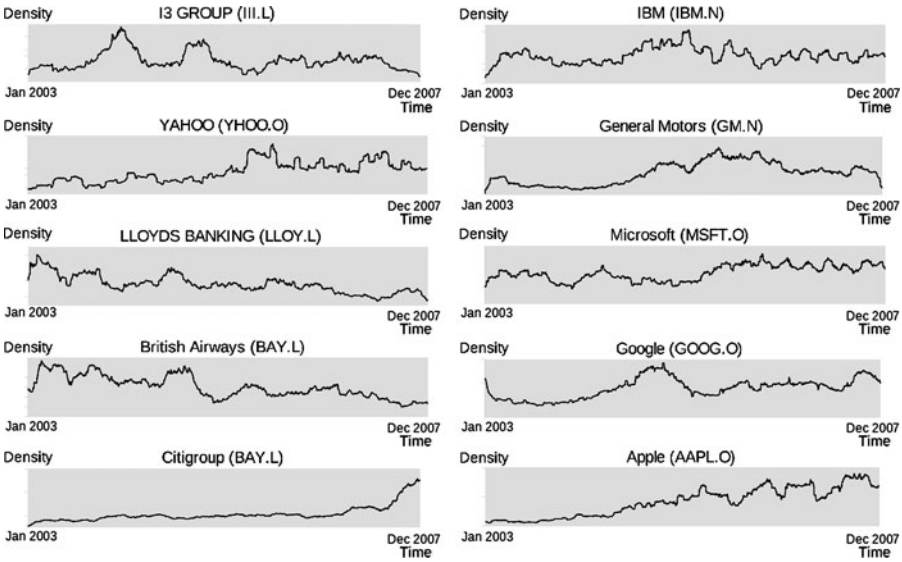

Fig. 1 Non-uniform data distribution for range queries from 2003-01-01 to 2007-12-31

regulators, and financial market participants across the globe. Every year up to 5 million additional news announcements are added to the existing SIRCA database. As of today, over 40 millions news announcements in more than 19 different languages have been captured, stored, and made available to researchers. To illustrate the non-uniform nature of this database, we define *data density* as the size of news announcements over a set period. Figure 1 plots some of the densities for news data related to several companies over a time interval of 5 years (2003 to 2007). In these figures, density has been defined over a weekly period.

For example, one can see that news announcements related to Apple (the company's tag—AAPL.O) tend to be more frequent during the period of 2006–2007 as compared to the period 2003–2005. Partitioning of a range query without any consideration to the distribution of data will lead to unbalanced query workload across database replicas. Therefore, this work proposes a density-driven query partitioning and load-balancing approach as described in the forthcoming section.

3 Proposed approach

The proposed density-based query partitioning mechanism allows the workload to be balanced among a set of database replicas deployed over a cloud infrastructure. Partitioning a range query $q = \{t_s,\ t_e, TS\}$ into two queries $q_1 = \{t_{s_1},\ t_{e_1}, TS\}$ and $q_2 = \{t_{s_2},\ t_{e_2}, TS\}$ is considered to be optimal if the estimated query processing times of both queries are as close as possible. We propose the following formulation to estimate the query processing time of a query q based on data density.

Definition 3 (Query processing time estimation)

$$\forall q = \{t_s,\ t_e, TS\}, \quad PTE(q) = \alpha \sum_{n \in RS(q)} s(n)$$

The estimated query processing time $PTE(q)$ is proportional to the size of the result set $RS(q)$. α is the coefficient and it depends on the number of VM instances that hosts the query processing service as well as the type of Relational DBMS (e.g. MySQL, PostgreSQL). In our experiments, we allocated small VM resource instances to MySQL relational DBMS. Changing the size of the VM resource instance to medium will have a clear impact on α. To conclude, the value of α can only be measured for various VM resource configurations. In this work, we hosted the query processing services (MySQL RDBMS) on homogeneous set of VMs having the same configuration (e.g. cores, speed, family, physical memory, etc.). As a result, the value of the parameter α was same for all VMs and hence was normalized to 1 during the query partitioning phase.

Based on the above formula, the partitioning of a query $q = \{t_s,\ t_e, TS\}$ into two equally load balanced queries $q_1 = \{t_{s_1},\ t_{e_1}, TS\}$ and $q_2 = \{t_{s_2},\ t_{e_2}, TS\}$ must correspond to a minimal value of

$$\left| \sum_{n \in RS(q_1)} s(n) - \sum_{m \in RS(q_2)} s(m) \right|$$

In order to optimally partition range queries, a solution should ensure that the sizes of result sets are similar. In that case, the range query partitioning based on density is defined as follows:

Definition 4 (Query partitioning based on density) The Query Partitioning Based on Density for k database replicas, noted $QPBD : (k, q) \longrightarrow \{q_1, \ldots, q_k\}$, ensures that:

$$\forall q = \{t_s,\ t_e, TS\}, \quad \forall q_k, q_p \in QPBD(k, q), S(RS(q_k)) \simeq S(RS(q_p)).$$

In other words, it ensure equality of sizes of the result sets of each query partition.

Such query partitioning is possible if the sizes of query results are predictable. This is made possible by pre-computing the density for a set period (referred to as the minimum density interval) of news articles for each company or topic and indexing them along with tagging information. The minimum density interval is set to one day in the experiments undertaken in this paper. The minimum density interval forms the basis for the query partitioner when determining sub-queries for a given range query. Depending on the frequency of news data items, the query partitioner could be configured to consider different minimum density interval. Clearly, this may have an impact on the number of sub-queries spanning out of a range query. For evaluating the proposed approach, any level of granularity (minute, hour, day, week, etc.) is acceptable as long as the pre-processing phase is applying the same minimum density interval. Since the news datasets that we are considering for this work span over a number of years, we perceived that setting the minimum density interval to one day is a reasonable assumption. Querying such an index allows the system to get the

Algorithm 1 for Query Partitioning Based on Density

```
INPUT k: number of partitions;
INPUT q: query to partition;
SET s = t_s(q);
SET Q: set of sub queries to construct;
FOR e BETWEEN t_s(q) AND t_e(q)
{
 IF (PTE({s, e, TS}) ≥ PTE(q)/k)
 {
   SET new query {s, e, TS} into Q;
   SET s = e;
 }
}
IF (s ≠ t_e(q))
 SET new query {s, t_e(q), TS} into Q;
RETURN Q;
```

sum of sizes of all news articles per day for a given query. We define such a density measure as follows.

Definition 5 (Density distribution) The density distribution of a query $q = \{t_s,\ t_e, TS\}$ is $DD(q) = \{S(RS(q_1), S(RS(q_2), \ldots, S(RS(q_n)\}$ where $\forall 1 < i < n\ /\ q_i = \{t_s + (i - 1) * (t_e - t_s)/n,\ t_s + (i) * (t_e - t_s)/n, TS\}$. In other words, if n is equal to the total number of days in the interval $[t_s, t_e]$, then $\forall 1 < i < n\ /\ q_i$ is the query q applied on a 1 single day. In such a case, the density distribution corresponds to the expected size of result sets of q for every day in the interval $[t_s, t_e]$. In this paper, we assume n to be automatically set to the total number of days in the interval $[t_s, t_e]$.

Knowing the size of news articles for a given query per day allows the design of an algorithm that splits a range query based on equality of result set sizes and operates in $O(n)$ (where n is the total number of days in the date range of a query). n is automatically computed by the algorithm from a given query's start and end date. Based on the size, different queries could lead to a search over a different number of days. For evaluation, query sizes (date ranges) are randomly chosen from the time interval [Jan. 2003, Dec. 2007].

Using a query partitioning technique based on density allows to reduce the number of density values to the number of days during the interval of a query. While experimenting with several years of news data, a query covering the entire interval of the data set covers at most 5 years $\times$ 365 days $= 1825$. The complexity of the splitting algorithm has a minor impact on the performance because as $n \leq 1825$, reducing the complexity of the splitting algorithm (e.g. to $O(\log\ n)$) will have a minor impact on the overall performance of the system.

The algorithm takes k (number of partitions) and q (original query to partition) as inputs and produces k sub-queries as detailed in Algorithm 1.

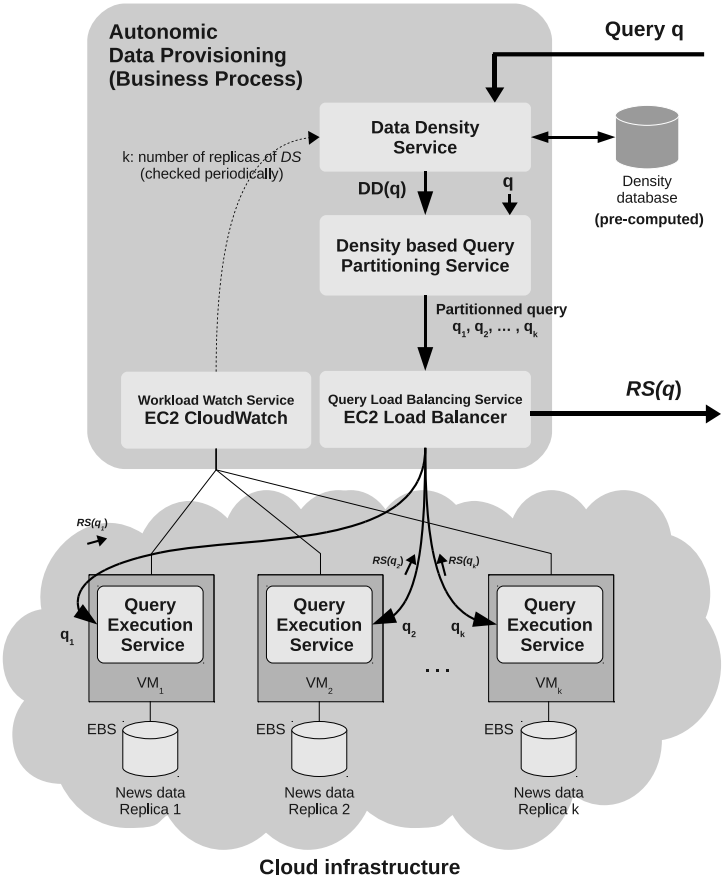

Fig. 2 Proposed SOA for query partitioning and execution on an Amazon EC2 infrastructure

To integrate the proposed solution with a cloud infrastructure, we propose a Service-Oriented Architecture (SOA) composed of the five main services illustrated in Fig. 2. We also show how the services can be supported by the Amazon Cloud infrastructure. The services are described as follows:

1. A Data Density Service: provides pre-computed density distribution helpful for the estimation of a query processing time. It is based on an inverted index of the density per day for every tag used in the news database. The Data Density Service's input is a range query $q = \{t_s,\ t_e, TS\}$ and its output is $DD(q)$ that is the size of news results per day over the range between t_s and t_e. Figure 1 has illustrated examples of the Data Density Service output.
2. A Density Based Query Partitioning Service: transforms a query q into a set of k queries $q_1, q_2, \ldots, q_k$ such that $PTE(q_1) \simeq PTE(q_2) \simeq \cdots \simeq PTE(q_k)$ and k is the number of available database replicas.
3. A Query Load Balancing Service: sends each q_i resulting from the Density based Query Partitioning Service on its corresponding replica of the data set. The exe-

cutions of all q_i are done in parallel. Then the results of each q_i are merged and the overall result of q is returned to the end user. In this work, we rely on the cloud-based load balancing service called Amazon Elastic Load Balancer (ELB). Such service plays the role of Query Load Balancing Service.
4. Query Execution Services: execute queries q_i resulting from the Density based Query Partitioning Service on database replicas assigned by the Query Load Balancing Service.
5. A Workload Watch Service: allows to monitor the workload of each virtual machine and provide a wide range of metrics useful for the load balancer to effectively distribute the workload across the available virtual machines. In this work, we rely on the cloud-based workload watching service provided by the EC2 Cloud called "Cloud Watch" which checks periodically the number of available replicas which can change due to the elasticity of the cloud.

A Business Process (BP) is responsible for orchestrating all the above services whenever a query is executed. This approach is very flexible as it allows the BP to be modified to accommodate a different cloud provider for example. The orchestrator is designed based on an extensible software engineering pattern that allows to plug-in service APIs of different clouds. Though our current implementation works with Amazon EC2 cloud, it can be extended to support other clouds including Microsoft Azure and GoGrid without much modifications to the design of the orchestrator. Recent developments in the cloud API domain such as Simple Cloud, Delta Cloud, JCloud, and Dasein Cloud abstracts APIs related to multiple cloud providers such as Amazon EC2, GoGrid, etc. Our future developments can leverage one of these APIs to enable orchestration of services across multiple clouds.

4 Experiments and validation

4.1 Experimental setting

Hardware and software configuration We ran our experiments on Amazon EC2 using standard small instances in US-west location. By default, a small instance has the following hardware configuration: 1.7 GB of main memory, 1 EC2 Compute Unit (i.e. 1 virtual core with 1 EC2 Compute Unit), 160 GB of local instance storage, and a 32-bit platform. Over the lifetime of the experiment, we varied the number of active instances over the range $[2, 16]$ in the step of 2^x where $x \in \{1, 2, 3, 4\}$. This was done in order to evaluate the performance of the system with varying query processing capabilities. For each instance type, we used an Ubuntu 10.04 operating system configured with a MySQL 5.1 database server. Each instance implements a 2-tier web application architecture (refer to Fig. 2), where an Apache web server with PHP modules forms the basis for the application tier, while the database tier is implemented using MySQL. The database tier stores a news dataset that requires 80 GB of storage space.

An Amazon's small instance has access to a local disk (secondary storage) which is not persistent by default. When an instance is shut down, both its state and contents on the local disk are purged. A cloud-based deployment model that cannot guarantee

data persistence is not a favorable option. To solve this, Cloud providers including Amazon and Microsoft Azure, offer *off instance storage* volumes that persist independently from the life on an Instance. These off instance storage volumes, which are referred to as the Elastic Block Store (EBS) in Amazon EC2 and XDrive in Microsoft Azure are particularly suited for applications that require database, file system, or access to raw level storage. The main advantages of architecting applications using off instance storage include: (i) each storage volume is automatically replicated, this prevents data loss due to failure of any single hardware component and (ii) storage volumes provide the ability to create point-in-time snapshots, which could be persisted to the cloud specific data repositories (such as Amazon's S3 and Azure Blob). Keeping the aforementioned advantages in mind, we mounted an EBS volume on the Amazon's S3 service for each of the instances that required a persistent news dataset.

Workload configuration The testing tool that we used to generate the query load was JMeter, which is a java desktop application designed to load test functional behavior and measure performance. It provides features that can be used to simulate a heavy load on a web server, network or an object to test its performance under varying load types. JMeter has a well developed graphical user interface which permits the visualization of test inputs and performance results. As an alternative to JMeter one could also use HTTPerf and ApacheBench for randomly generating workload. An architectural diagram of the test setup is shown in Fig. 2. JMeter generates the query workload based on a Poisson distribution, whose mean inter-arrival delay was set to 10 *seconds*. The queries are directly submitted to the Amazon's ELB service. The company id (RIC) and the date range are chosen randomly from the available companies list and period (Jan. 2003 – Dec. 2007). A sample query in our experiments had the following semantics: $q =$ "search news announcements with $TS(q) =$ AAPL.O (related to Apple) starting date = 01/01/2003 and end date = 01/01/2004".

As the ELB is not launched as an instance, we will address it as an architectural component as opposed to a hardware or software in the following discussions. Amazon does not currently support different instance sizes for hosting ELBS, so all tests were run with the standard ELB. Further, no performance tuning or configuration is currently possible on ELBs. The only configuration that we set with regards to the ELB was that only a single availability zone (US-west) was enabled for load distribution. The instances hosting the web server and the database server were configured to register automatically with the ELB during their start up phase.

The experiment has the following query workflow: JMeter directly submits the queries to an ELB; the ELB dispatches the query to one of the known query service instances in cloud; and upon successful execution of the query, the results are directly returned to the JMeter, where they are logged for further analysis. The performance metrics collected and analyzed in all tests was the Disk I/O read throughput and the average query processing time. The *Read Throughput* is the key metric in database applications as it may require instances to store immediate results on local disks or request data from mounted storage volume (EBS) if the data is not cached locally. The *Average Query Processing Time* is the delay between the time-stamp when the JMeter submits a news search query to Amazon's ELB and the time-stamp when the JMeter receives a successful reply from the query execution service. This information

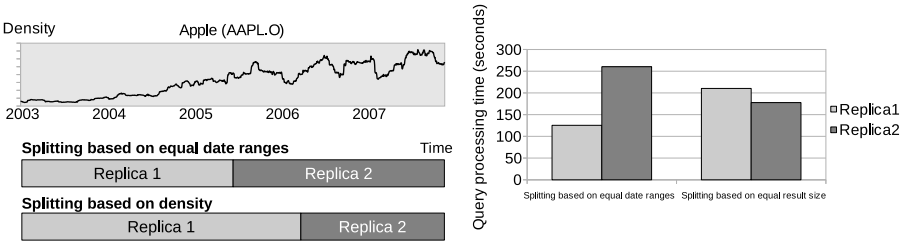

Fig. 3 Performance of built-in load-balancing capability of database systems

is continuously logged with a special result analysis service over the life-span of the tests. We ran our experiments with the following objectives in mind: to measure the efficiency of workload-aware query load-partitioning and analyze the impact it may have on real applications (news query service). We also conducted experiments to compare the efficiency of the proposed approach against a classical one (existing built-in load-balancing capability of database systems).

4.2 Evaluating built-in load-balancing capability of database systems

Our work is motivated by the fact that classical approaches based on existing DBMS' built-in facilities for partitioning range queries are unable to perform optimal balancing of heterogeneous workload. To demonstrate this statement, a simple test is conducted that partitions a range query based on equality of ranges. The range query with the following configuration is injected into the system: $q =$ "search news announcements with $TS(q) =$ AAPL.O (related to Apple) where start date $=$ 01/01/2003 and end date $=$ 31/12/2007 (total search interval $=$ 5 years). Since there are two replicas of the query execution service, the search query is split into two sub-queries with each requesting a news dataset over 2.5 years. Figure 3 shows the results of this test. Confirming our anticipation, the results show that uniform partitioning of queries across the service replicas without considering the underlying data density distribution leads to unbalanced workload. Recall that such behavior exists due to the intrinsic nature of the news dataset, where the queries searching over a similar space (number of months), may have different amount of data to index. To further push our argument, Fig. 3 also plots the performance gain achieved when the partitioning and routing of a query is done based on the proposed density-based approach.

4.3 Evaluating the performance of cloud-based deployments

So far, we ran experiments for studying the classical query load-balancing approach, which is intrinsic to database system implementations. Thus, the natural next steps are to analyze

(1) whether the performance variability and degradation observed in previous sections will be efficiently handled through the cloud-based deployment of query execution service and
(2) to which extent the load-balancing efficiency across service replicas is improved by applying the proposed density driven approach.

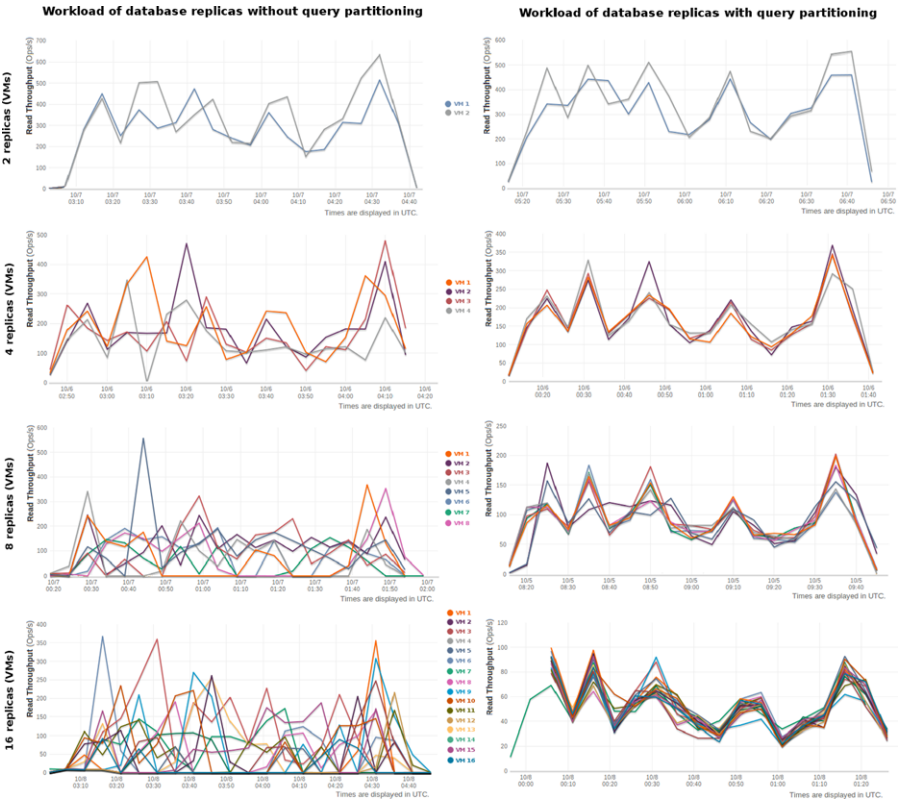

Fig. 4 Workload of database replicas (virtual machines) for 2, 4, 8, and 16 replicas with (*right*) and without (*left*) query partitioning

In these experiments, a query workload (search interval) is generated, which contains queries of various sizes for randomly selected companies. A total of 500 random queries are included as part of the workload and sent to the cloud-deployed service instances at a rate of 6 queries per minute. It is worth noting that this rate is exactly same as the original query rate experienced by SIRCA's query execution service (an estimated 1 million queries per year). The experiment lasted 82 minutes and is being done to study two distinct cases: the first one partitions queries based on the news data density and the second one does uniform portioning (i.e. without considering the data density).

The results pertaining to the read-throughput of instances are shown in Fig. 4.

4.4 Limitations

The first drawback of the proposed approach is the required pre-processing phase to build the data density database which can be time consuming. However, such a density database is only built once as the news dataset is static and no updates to the density database are required afterwards. The pre-processing phase allows an index to be built. The processing time is obviously high in this phase and depends on the

type of DBMS and the type of built-in index used. In our experiments, indexing the entire dataset took 23 hours on a small instance. Increasing the performance of this phase can be done by executing it on a much bigger instance. However, this phase is only done once because the news data is historical and never updated. This could be a limitation when using the proposed approach on a frequently updated news database.

Once the density database is created, querying it is in $O(\log n)$ where n is the total number of tag-day (a tag-day is the unique association of a tag and a day during which the tag has been used). That is made possible by using a b-tree index on dates and a hash index on tags for the density data. Experiments show that querying the density of a query requires an average of 0.1 seconds when using a 5-year dataset corresponding to over 27 million tag-days in the density database.

The second drawback is that the number k of queries resulting from the partitioning of a query q is limited to the number of days composing the date/time range of q. For example, a query on news between 01/12/2003 and 03/12/2003 could only be distributed on 3 replicas, one day each. This is due to the fact that an indivisible date range has to be defined (which is 1 day in our case) to allow the construction of an efficient data density index. Therefore, a data set of 5 years of news data would allow no more than 1825 partitions for any query. This limitation can be overcome by choosing a smaller time interval (e.g. an hour) but this would significantly increase the time required to build the index.

5 Related work

In this section, we look at the current state-of-the-art and compare it against the work proposed in this paper.

When considering cloud database services, cloud-based relational database services such as Amazon RDS and Microsoft SQL Azure have been introduced to the market. However, existing offerings have severely limited abilities in their support for workload-aware query partitioning and load-balancing. A recent approach [16] has considered a query partitioning strategy, but is only applicable to OLTP and web workloads. Since OLTP/web workloads are characterized by short-lived queries with little internal parallelism, the approach is not directly applicable to our case where sub-queries can be parallelized across replicated database service instances. There has been a substantial amount of work [17] on the problem of tuning virtual machine configurations for database workloads and on the problem of making database systems adaptable to hardware resource allocation [21, 22]. However, in this paper, we are fine tuning the partitioning of queries to better balance the load across replicated database service instances.

In the area of data Clustering, clustering records [23–25] is of particular interest for range query processing. The approach involves physically placing records having a common characteristic onto the same cluster. In the case of our news data, clustering can be based on date ranges and/or tags. Clustering based on date ranges allows to speedup a range query processing if the date range is short (few days or weeks) but results in poor performance with larger date ranges. Furthermore, news data are usually tagged with several tags and, therefore, applying clustering on tags could lead

to duplication of news item across clusters, hence increasing the size of the database. Moreover, the size of the clusters can significantly vary as some tags are used more frequently than others. Finally, combining clustering based on date range and tags would inherit the disadvantages of both solutions. For the above reasons, clustering based on date ranges and/or tags is not suitable approach for the type of data set we are dealing with, i.e. SIRCA's news data.

Another class of approach includes replicating database instances to distribute range query processing. In [13], the authors address the two conflicting problems of ensuring data-access load balancing and efficiently processing range queries over peer-to-peer networks. A novel hash function is proposed which (a) preserves the ordering of data to ensure efficient range query processing, and, (b) replicates and fairly distributes popular data or/and ranges among peers. Although such an approach ensures load balancing querying of "popular" data (frequently requested), it performs poorly for infrequently accessed data sets.

6 Conclusion & future work

In this paper, we introduced a hybrid workload-aware approach for efficiently balancing workload (range queries) across relational database-as-a-service for cloud computing environments. The proposed approach overcomes two significant challenges: efficient query partitioning and load-balanced workload placement. For query partitioning, we developed a novel data density distribution estimation algorithm. For workload placement, we implemented the query partitioning algorithm as an additional service on top of existing cloud-based load-balancer services such as Amazon EC2 Loadbalancer. Our approach allows a minimum density interval to be configured according to granularity of relational dataset and composition of search queries. Based on our evaluation results, we believe that workload-aware database-as-a-service can be made prime time and further research and development work is needed in this domain.

As future work, we will develop query workload and relational database performance models that can detect changes in workload intensity which may occur over time and allocate or de-allocate database instances accordingly in order to achieve performance targets. We will develop new workload prediction models that build upon the recent advances in Computational Statistics (CS) techniques, and allow the capture of the performance behavior from the actual traces of incoming requests. To this end, we will build upon existing CS techniques including kernel canonical correlation analysis and quadrative response surface model. We will investigate a flexible, layered queueing model, to determine how much hardware resources (computing, storage, and network) to provision for a database-as-a-service application. The goal is to allocate sufficient capacity to each layer (load-balancer, business logic, and database) so that its performance target can be met under varying workload conditions within a minimum possible cost. Finally, we would look into the cloud-agnostic orchestration environment which can be transformed into cloud-specific configurations and used for facilitating deployment across multiple clouds.

Acknowledgement We wish to acknowledge the Smart Services CRC in Australia for sponsoring this research project and SIRCA for providing access to its news database. During the initial phases of the research (experiments and manuscript preparation), Dr. Rajiv Ranjan was employed at the University of New South Wales.

References

1. Armbrust M et al (2009) Above the clouds: A Berkeley view of cloud computing. Tech Rep UCB/EECS-2009-28, EECS Department. University of California, Berkeley
2. Rochwerger B et al (2009) The RESERVOIR model and architecture for open federated cloud computing. IBM J Res Dev 53(4):535–545
3. Buyya R et al (2009) Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Gener Comput Syst 25(6):599–616
4. Gillett FE et al (2008) Future view: The new tech ecosystems of cloud, cloud services, and cloud computing, Tech rep, Forrester Research, Inc
5. Varia J (2009) Cloud architectures, Tech rep, Amazon Web Services
6. Windows azure platform. http://www.microsoft.com/azure/ (accessed August 2011)
7. Wang L et al (2010) Provide virtual machine information for grid computing. IEEE Trans Syst Man Cybern, Part A, Syst Hum 40(6):1362–1374
8. Amazon cloudwatch service. http://aws.amazon.com/cloudwatch/ (accessed August 2011)
9. Amazon load balancer service. http://aws.amazon.com/elasticloadbalancing/ (accessed August 2011)
10. Amazon elastic mapreduce service. http://aws.amazon.com/elasticmapreduce/ (accessed August 2011)
11. Force.com cloud solutions (saas). http://www.salesforce.com/platform/ (accessed August 2011)
12. Wang L et al (2010) Cloud computing: a perspective study. New Gener Comput 28(2):137–146
13. Pitoura T et al (2006) Replication, load balancing and efficient range query processing in dhts. In: Advances in database technology - EDBT 2006, vol 3896. Springer, Berlin, pp 131–148
14. Chen D et al (2010) Synchronization in federation community networks. J Parallel Distrib Comput 70(2):144–159
15. Olofson CW (August 2010) Keeping your data in the clouds and your feet on the ground, whitepaper, idc, sponsored by: Sybase
16. Curino C et al (2011) Relational cloud: A database service for the cloud. In: 5th biennial conference on innovative data Systems research. Asilomar, CA
17. S A et al (2008) Automatic virtual machine configuration for database workloads. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, Vancouver, Canada. pp 953–966
18. Sakr S et al (2011) A survey of large scale data management approaches in cloud environments. IEEE Commun Surv Tutor PP(99):1–26
19. Brantner M et al (2008) Building a database on s3. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data. ACM, Vancouver, pp 251–264
20. SIRCA, Thomson Reuters news database. http://www.sirca.org.au/ (accessed august 2011)
21. S J et al (2006) Adaptive self-tuning memory in db2. In: Proceedings of the 2006 (32nd) international conference on very large data bases, VLDB Endowment. pp 1081–1092
22. Narayanan D et al (2005) Continuous resource monitoring for self-predicting dbms. In: Proceedings of the 2005 IEEE international symposium on modeling, analysis, and simulation of computer and telecommunication Systems. IEEE Press, New York
23. C CY et al (1993) Optimal mmi file systems for orthogonal range retrieval. Inf Syst 18(1):37–54
24. Harris P et al (1993) Optimal dynamic multi-attribute hashing for range queries. BIT Numer Math 33(4):561–579
25. Lee J et al (1997) A region splitting strategy for physical database design of multidimensional file organizations. In: Proceedings of the 1997 (23rd) international conference on very large data bases. Kaufmann, San Francisco, pp 416–425