EdgeX: Edge Replication for Web Applications

Hemant Saxena
University of Waterloo
h2saxena@uwaterloo.ca

Kenneth Salem
University of Waterloo
kmsalem@uwaterloo.ca

***Abstract*—Global Web applications face the problem of high network latency due to their need to communicate with distant data centers. Many applications use edge networks for caching images, CSS, javascript, and other static content in order to avoid some of this network latency. However, for updates and for anything other than static content, communication with the data center is still required, and can dominate application request latencies. One way to address this problem is to push more of the web application, as well the database on which it depends, from the remote data center towards the edge of the network. In this paper, we present preliminary work in this direction. Specifically, we present an edge-aware dynamic data replication architecture for relational database systems supporting web applications. Our objective is to allow dynamic content to be served from the edge of the network, with low latency.**

I. Introduction

Achieving low latency for web-based applications is an on-going challenge. A study by Akamai [1] reports that "57 percent of online shoppers will wait three seconds or less before abandoning" a site. For Amazon, a delay of 100ms costs 1% of sales [2]. A key source of latency for web applications is what Leighton refers to as the *middle mile*, i.e., the need to transmit information between the origin data center and the end user. [3]

A widely-used technique for reducing web application latency is to cache information close to the end user [4]. Applications use content delivery networks (CDNs), like Akamai's, for caching static content, such as images, CSS, and javascript files. Although such edge caching is clearly beneficial, it does not completely solve the latency problem. Even with CDNs, there is typically at least one round trip to the origin data center for each application request. In addition, CDNs become less effective when most of the application's data are frequently updated or user-generated. Figure 1 shows a waterfall diagram of a request to view an item at $eBay.com$. This request triggers a number of requests to content delivery networks (CDNs) to retrieve static content like images and CSS files cached at the edge of the network. However, the initial request goes all the way to a remote data center to obtain item pricing, quantities, and other dynamic content. The latency of this request is an order of magnitude higher than that of the CDN requests, and dominates the overall request latency.

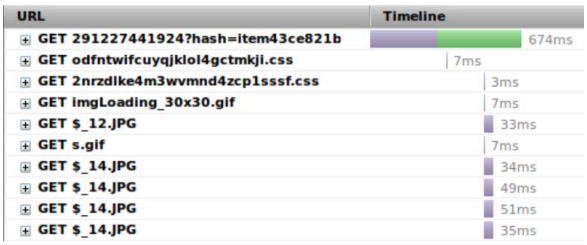

Figure 1: Request latencies for $ebay.com$

In this paper, we consider a technique for going beyond edge caching to reduce request latencies for web applications, and in particular for web applications that make heavy use of dynamic, user-generated content. Our approach allows edge servers to do more than just cache static content. Instead, we allow portions of the web application itself to run at the edge of the network, so that requests can potentially by handled entirely at the edge, avoiding the middle mile entirely. The objective of this technique is to *reduce web application latency by taking advantage of geo-locality*, allowing application operations to be executed at edge sites close to the application's end users.

Since web applications typically depend on a backend database, handling application requests at the edge also requires that this database, or at least parts of it, be present at the edge of the network as well. To accommodate this, our approach allows the application's database to be partially replicated at the edge of the network. How to manage such *edge replication* is the focus of this paper.

Our approach is implemented in a prototype system called EdgeX. EdgeX uses a novel two-part approach to manage replication, e.g, to decide which portions of the database should be replicated at each edge site. The first part of EdgeX's approach is a static analysis of the application's data access and consistency requirements. The purpose of this analysis is to determine which portions of the database can be safely replicated at the edge, i.e., can be replicated without violating consistency requirements. This analysis labels each portion of the database as either *edgeable* or not. Non-edgeable parts of the database are stored only at the origin data center, and not at the edge. The second part of EdgeX's approach is a run-time mechanism that determines *whether*, *when*, and *where* to replicate the edgeable parts of

2159-6190/15 $31.00 © 2015 IEEE
DOI 10.1109/CLOUD.2015.147

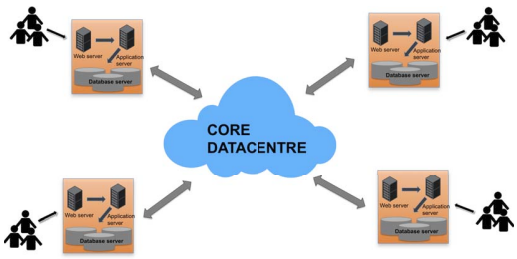

Figure 2: System architecture, core site with four edge sites.

ITEMS

id	quantity	reserve_price	nb_of_bids	max_bid	seller	category
3	1	5302	12	4853	993038	3

BIDS

id	user_id	item_id	qty	bid	max_bid	date
23	667382	3	1	4771	4776	2014-10-10 09:48:09
24	842569	3	1	4778	4785	2014-10-10 09:48:09
25	350294	3	1	4787	4796	2014-10-10 09:48:09
26	466678	3	1	4793	4799	2014-10-10 09:48:09
27	674629	3	1	4802	4811	2014-10-10 09:48:09
28	969515	3	1	4809	4816	2014-10-10 09:48:10
29	756439	3	1	4812	4815	2014-10-10 09:48:10

Figure 3: A partition of `ITEMS` cluster.

the database. EdgeX's run-time mechanism seeks to exploit *geo-localization* in the application's request patterns. That is, data that are used primarily by end-users in a particular geographic area can be replicated to a network edge site in the same geographic area.

II. System Overview

We assume a *two-tier infrastructure* consisting of the origin data center, which we refer to as the *core*, and multiple *edge* sites, as shown in Figure 2. As is the case for CDNs, we assume that network latencies between end users and nearby edge sites are lower than network latencies between end users and the core. Unlike the edge sites in CDNs, which serve as caches, we assume that edge sites are capable of running the web application's server-side code, and of storing and managing application data.

We assume that an application uses a relational database whose schema can be partitioned into hierarchically organized *clusters* of tables, much as in Google's Spanner [5]. Within each cluster, one table is designated as the *parent* table, and all other tables in the cluster are linked to the parent, directly or indirectly, via foreign key relationships. For example, Figure 3 shows an example of a cluster consisting of `ITEMS` and `BIDS` tables. `ITEMS` is the parent table to the `BIDS` table via a foreign key (`item_id`).

Within each cluster, we define *partitions* based on the values of the primary key of the parent table. For each primary key value x, the partition consists of the parent table tuple with that key, along with all tuples from the other cluster tables that link, directly or indirectly, to that parent tuple through the foreign key relationships. For example, the cluster shown in Figure 3 is partitioned by `item_id`. Each partition consists of a single item from the `ITEMS` table, along with the bids for that item from the `BIDS` table.

In EdgeX, partitions (or entire clusters) are the units of replication and distribution. There can be one or more replicas of each partition distributed among the core and edge sites. The EdgeX system does not determine how best to cluster the database schema. Rather, it assumes that the clusters are given. Clusters can be defined by application developers, as in Spanner, or chosen automatically [6].

EdgeX uses partition-level mastership, similar to record-level mastership in PNUTS [7], to manage the synchronization of partition replicas. Each partition has a single master copy, which is located at a site chosen by EdgeX. Each partition may have zero or more secondary copies, also placed by EdgeX. All partition updates are performed at the partition's primary site. EdgeX then propagates these updates lazily to the remaining replicas.

It is assumed that the application consists of a set of parameterized, pre-defined server-side operations. End-user application request for these operations are directed to the nearest edge site. As we will describe shortly, that edge site may or may not be capable of handling the request locally, using locally available data. If the request can be handled by the edge site, it handles the request and responds, avoiding the need for any round trips to the core. If the edge cannot handle the request locally, it acts as a proxy and forwards the request to the core, which handles the request as usual. Thus, in EdgeX, each operation is executed at a single site - either an edge or the core.

III. Data Replication

EdgeX chooses where to place primary and secondary copies of each database partition. By monitoring the usage of database partitions, it can move partition replicas to edge sites in geographic locations where the partition is heavily used. EdgeX is constrained in several ways in its choice of where to place the primary and secondary copies of each database partition. First, EdgeX must ensure that it is possible to execute each application operation at a single execution site: either at an edge, or at the core. Second, EdgeX must ensure that application-specified database consistency requirements are satisfied. This is important because EdgeX synchronizes partition replicas lazily, making secondary copies stale relative to the primary.

EdgeX uses a novel two-part approach to solve this constrained data placement problem. First, EdgeX performs a static analysis of the application's consistency requirements to associate a *replication policy* with each database cluster. A cluster's replication policy guides and constrains where the run-time system is permitted to place replicas of that cluster's partitions. Second, at run-time, EdgeX determines the placement of partition replicas based on observed data access patterns, within the guidelines imposed by the statically-determined replication policy for each cluster. EdgeX's replication policies are such that the run-time system is free to make placement decisions for each partition

independently of the current placement of other partitions. This significantly simplifies the task of the run-time system. We describe the replication policies and the static analysis that produces them in the remainder of this section, and present the run-time system in more detail in Section IV.

A. Static Application Analysis

EdgeX assumes that the application's server-side code implements a set of parameterized request types $\{R_1, ..., R_n\}$. Handling each type of request may require access to one or more of the database clusters.

EdgeX's static analysis takes as input database access information and consistency requirements for each type of application request. This information must be provided by the application developer. Specifically, for each request R_i, the analysis is given:

- the set of database clusters that requests of type R_i may access, and
- for each such cluster, a flag indicating whether R_i's access to that cluster can be localized to a single partition, and
- for each cluster, a *consistency constraint* indicating whether R_i requires access to an up-to-date copy of the cluster, or can tolerate access to a stale copy.

We say that R_i requires *single-partition* access to a cluster C if it can be determined that each request of type R_i will read and/or write data from a single partition of C, and that partition can be determined based on the value of a request parameter. For example, if C contains information about users and is partitioned by `UserID`, R_i requires single-partition access to C if R_i takes a `UserID` as a parameter and only reads or writes data in that user's partition of C. Different instances of R_i may access data about different users, but each instance must be known to access data about a single user. Request types that are not single partition are called *multi-partition*.

The consistency constraint for R_i on cluster C indicates whether R_i requires access to an up-to-date copy of C (*fresh* access), or whether it can tolerate a stale view of C (*stale* access). At run-time, EdgeX's replica synchronization mechanism ensures that all updates to a cluster are serialized by the primary copy of that cluster, and then propagated lazily and in order to any secondary copies. If R_i requires fresh access to C, then it must run where the primary copy of C is located. Otherwise, R_i is assumed to be able to tolerate the potentially stale view of C that would be found at the location of a secondary copy. Any request type that updates a cluster requires *fresh* access.

Given this input for all request types, the primary task of the static analyzer is to choose a run time *replication policy* for each cluster. EdgeX supports two possible replication policies, which we refer to as *Core/Static* and *Edge/Dynamic*. The Edge/Dynamic policy for cluster C indicates that EdgeX's run-time system is free to place the primary copy of each of C's partitions at an edge site or at the core, at its discretion, and regardless of the placement of C's other partitions. The run-time is also free to place secondary copies of C's partitions anywhere, subject to the constraint that a copy (either primary or secondary) of each partition must be present in the core. In contrast, the Core/Static policy indicates that the run-time system must keep the primary copy of all of C's partitions at the core site, and that it should place a secondary copy of C's partitions at every edge site. This placement is fixed and will not change at runtime.

Clearly, assigning the Edge/Dynamic policy to clusters is desirable, as it gives the run-time system the freedom to exploit geo-localized access patterns by moving the primary copies of partitions to edge sites. However, the static analysis is constrained in its policy assignments by two factors. First, it must be possible to execute each application request at a *single site* (edge or core) in the system. Second, the consistency requirements specified for each request type must be observed. To understand these constraints' effects on policy assignment, consider the following example cases:

Case 1: Suppose request type R_i requires fresh, multi-partition access to some cluster C. In this case, the analysis *must* assign the Core/Static policy to C. If the Edge/Dynamic policy were assigned instead, the run-time system would be free to scatter the primary copies of C's partitions across different sites. Thus, there may be no single site in the system at which requests of type R_i could run.

Case 2: Suppose that request type R_i requires fresh, single-partition access to cluster C_j and stale single-partition access to cluster C_k. In this case, the analysis has two choices. It can assign Edge/Dynamic to C_j and Core/Static to C_k, which will allow each R_i request to run at whichever site holds the primary copy of its target partition in C_j, as determined by the run-time system. Alternatively, the analysis can assign Core/Static to both C_j and C_k, which will ensure that requests of type R_i can run at the core.

Using similar reasoning, we can generate a (potentially disjunctive) policy constraint corresponding to each consistency requirement specified by the application. The analysis then chooses policy assignments that satisfy all such constraints. Because of the nature of our policies and consistency requirements, there will always be at least one policy assignment that will satisfy all of the constraints, namely assigning the Core/Static policy for all clusters. This is an undesirable assignment as it leaves the run-time system without the flexibility to exploit the edge sites, but it does ensure that the application will be able to run the same way that it would have if there were no edge sites. In this sense, EdgeX's replication policies and static analysis can be viewed as a way of determining how much flexibility an application affords for edge execution, and then exposing that flexibility to the run-time system.

IV. Run-time System

EdgeX's run-time consists of a web server and relational database system at the core and at each edge site. Implementations of each of the application's web requests are present at the core and at each edge. To this basic infrastructure, EdgeX adds a mechanism for propagating database updates, and a mechanism for managing the placement of primary and secondary copies of database partitions at various sites.

A. Request Handling

A request issued by a client is sent to the nearest edge site, which determines whether the request can execute locally, at the edge, or whether it must be forwarded to the core for execution there. This decision is made based on whether data required by the incoming request are present at the edge site. To support these routing decisions, EdgeX maintains metadata at each edge site to indicate which cluster partitions are present in the edge site's database. These metadata are updated by EdgeXs partition placement manager when partitions replicas are added to or removed from the edge database. Two partition lists are maintained, one to indicate which partitions have their primary replicas at the site, and the other to indicate which partitions have a secondary replica there. In our EdgeX prototype, request forwarding is implemented using the Apache web server's *mod_rewrite* module, which performs rule-based rewriting and forwarding of request URLs. The *mod_rewrite* module parses the request URL and extracts the key of the required partition from the request parameters. It then checks the EdgeX metadata for this partition.

B. Update Propagation

Each partition has single master copy in the local database at one site, and zero or more secondary replicas in databases at other sites. Execution policies determined by EdgeX's static analysis, together with its request handling mechanism, ensure that all updates to a partition will be performed at the partition's primary site.

EdgeX uses a publish-subscribe mechanism to propagate database updates lazily among the databases at the core and edge sites. EdgeX defines a publication channel for each database partition, and each site subscribes to the channels for partitions it holds secondary copies of. Each site is responsible for pushing updates of all primary partitions it holds though the appropriate channels.

C. Replica Placement

EdgeX's run-time manages the placement of partition replicas for all clusters labeled as Edge/Dynamic by the policy analysis. Intuitively, the goal of EdgeX's replica placement manager is to replicate a partition to an edge site if most of the application requests that use that partition originate from that site. For example, in an classified advertising application like Kijiji or Craigslist, items sold by sellers from the Toronto region may be most frequently accessed by the buyers in the same region, and EdgeX can move data about those items to an edge site in the same region. If most updates for a partition originate at a site, EdgeX can move the primary copy of the partition there.

EdgeX has a parameter which limits the number of replicas of each partition, so that the cost of maintaining replicas can be controlled. The run-time system places the partition replicas such that the latency benefit is maximized. EdgeX has a central replica placement manager which periodically analyzes the workload of each edge site and decides whether and how the replica placement needs to be changed.

V. Results and Conclusion

We are currently in the process of evaluating EdgeX's ability to reduce web application request latencies. As an initial test, we applied EdgeX's static analysis to RUBiS [8], an auction site benchmark modeled after eBay. The resulting policy assignments allow up to 70% of RUBiS application requests to be handled entirely at edge sites. Thus, EdgeX is a promising mechanism for going beyond static caching at the network edge. The actual percentage of requests that the EdgeX run-time will handle at edge sites will depend on the amount of geo-locality present in the workload, but the policy assignments give an upper bound on the amount of "edging" that is possible for this workload.

VI. Acknowledgement

This research has been funded in part or completely by the Smart Applications on Virtual Infrastructure (SAVI) project under National Sciences and Research Council of Canada (NSERC) Strategic Networks grant number NETGP397724-10.

References

[1] *June 14, 2010 - New Study Reveals the Impact of Travel Site Performance on Consumers*, http://www.akamai.com/html/about/press/releases/2010/press_061410.html.

[2] R. Kohavi and R. Longbotham, "Online experiments: Lessons learned," *Computer*, vol. 40, no. 9, pp. 103–105, 2007.

[3] T. Leighton, "Improving performance on the internet," *Commun. ACM*, vol. 52, no. 2, pp. 44–51, Feb. 2009.

[4] J. Dilley *et al.*, "Globally distributed content delivery," *IEEE Internet Computing*, vol. 6, no. 5, pp. 50–58, Sep. 2002.

[5] J. C. Corbett *et al.*, "Spanner: Google's globally-distributed database," in *OSDI*, Oct. 2012, pp. 261–264.

[6] K. Q. Tran *et al.*, "JECB: A join-extension, code-based approach to OLTP data partitioning," in *Proc. ACM SIGMOD*, 2014, pp. 39–50.

[7] B. F. Cooper *et al.*, "PNUTs: Yahoo!'s hosted data serving platform," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1277–1288, Aug. 2008.

[8] E. Cecchet *et al.*, "Performance and scalability of EJB applications," in *Proc. ACM OOPSLA*, 2002, pp. 246–261.