EdgeX: Edge Replication for Web Applications

by

Hemant Saxena

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2015

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. This thesis presents preliminary work in this direction. Specifically, it presents an edge-aware dynamic data replication architecture for relational database systems supporting web applications. The objective is to allow dynamic content to be served from the edge of the network, with low latency.

Acknowledgements

I am extremely grateful to my supervisor Ken Salem for his guidance, support, and dedication throughout this thesis and during my graduate studies. His training and enthusiasm towards addressing challenging problems has had a positive effect in my life. I would like to thank my thesis readers, Tamer Ozsu and Khuzaima Daudjee, for their valuable comments.

I feel extremely privileged and humbled to have interacted with some amazing instructors, academicians, and fellow students during my time here at the University of Waterloo. Thanks to my fellow Waterluvians: Anirudh, Siddharth, Bharathwaj, Anup, Neeraj, David, Ankit and Vijay who kept me motivated by always joining me for Burrito Boyz dinner. In addition to this, I would like to say a special thanks to an amazing friend, who was also my hiking, running, biking and a patient dance partner for the last few months of my Master's program, thank you Shreya for being there.

Finally, I would like to thank my parents and brother for their unconditional love and support towards my academic career.

Dedication

This thesis is dedicated to my lovely parents.

Table of Contents

List of Tables

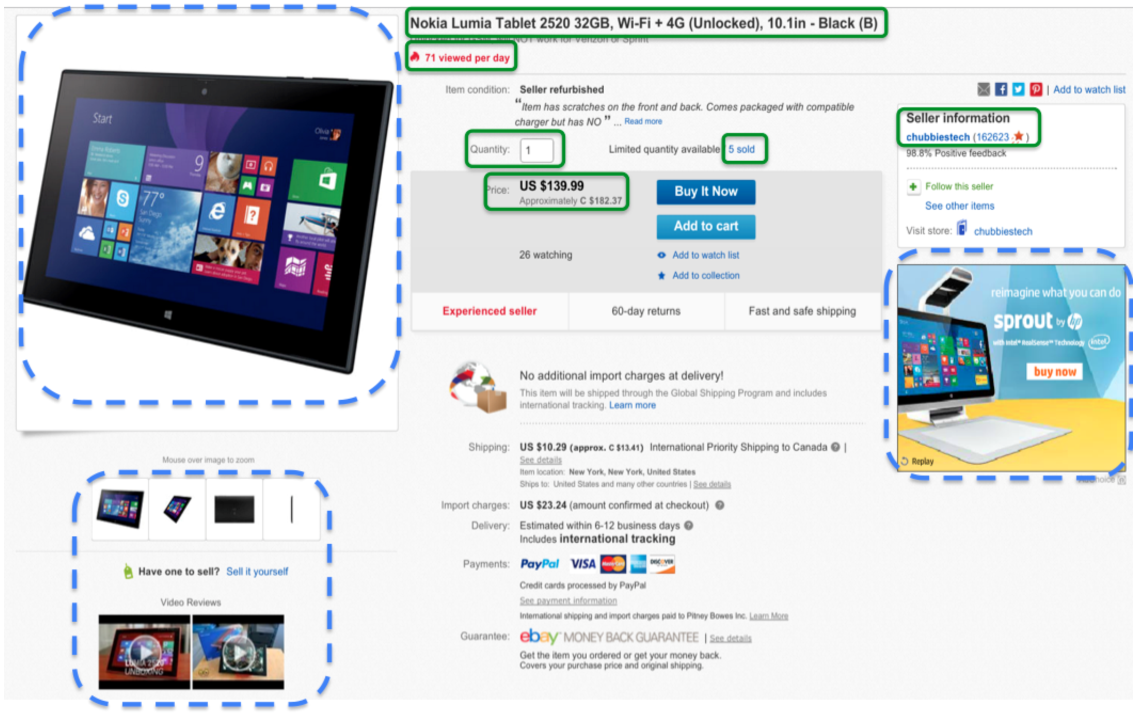
List of Figures

Chapter 1

Introduction

Achieving low latency for web-based applications is an on-going challenge. A study by Akamai [7] 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 [34]. 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 centre and the end user [37].

A widely-used technique for reducing web application latency is to cache information close to the end user [29]. Applications use content delivery networks (CDNs), like Akamai's, for caching static content, such as images, style sheets, 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 centre for each application request. In addition, CDNs become less effective when most of the application's data are frequently updated or user-generated, i.e., dynamic in nature. A typical web page displaying an item on *eBay* website consists of both static content and dynamic (user generated) content. Figure 1.1 shows some of the static content and dynamic content, which together forms a complete webpage. The content within the blue dashed boxes is the static content, mostly images, and the content within the green boxes is the dynamic content that is obtained from the database, mostly item details. To display a webpage, the application issues multiple sub-requests to fetch these individual contents. Figure 1.2 shows a waterfall diagram of the parent request to view an item at *eBay.com*. This request triggers a number of sub-requests to content delivery networks (CDNs) to retrieve static content like images and style sheets cached at the edge of the network. However, the first sub-request goes all the way to a remote data centre 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.



Nokia Lumia Tablet 2520 32GB, Wi-Fi + 4G (Unlocked), 10.1in - Black (B)

71 viewed per day

Item condition: **Seller refurbished**

"*Item has scratches on the front and back. Comes packaged with compatible charger but has NO*" ... Read more

Quantity: 1 Limited quantity available 5 sold

Price: **US $139.99** Approximately C $182.37

Buy It Now

Add to cart

26 watching

Add to watch list

Add to collection

Experienced seller | 60-day returns | Fast and safe shipping

No additional import charges at delivery!

This item will be shipped through the Global Shipping Program and includes international tracking. Learn more

Shipping: **US $10.29 (approx. C $13.41)** International Priority Shipping to Canada | See details

Item location: New York, New York, United States

Ships to: United States and many other countries | See details

Import charges: **US $23.24** (amount confirmed at checkout)

Delivery: Estimated within 6-12 business days

Includes **international tracking**

Payments: PayPal VISA

Credit cards processed by PayPal

See payment information

International shipping and import charges paid to Pitney Bowes Inc. Learn More

Guarantee: ebay MONEY BACK GUARANTEE | See details

Get the item you ordered or get your money back.

Covers your purchase price and original shipping.

Add to watch list

Seller information

chubbiestech (162623)

98.8% Positive feedback

Follow this seller

See other items

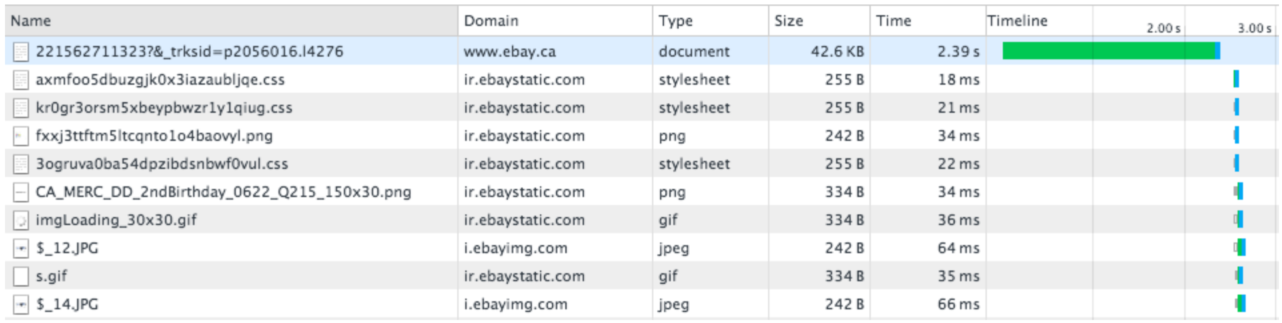
Visit store: chubbiestech

Mouse over image to zoom

Have one to sell? Sell it yourself

Video Reviews

Figure 1.1: Sample web page from *ebay.com*. Blue dashed boxes show static content and green boxes show dynamic content.



Name	Domain	Type	Size	Time	Timeline 2.00 s 3.00 s
221562711323?&_trksid=p2056016.l4276	www.ebay.ca	document	42.6 KB	2.39 s	
axmfoo5dbuzgjk0x3iazaubljqe.css	ir.ebaystatic.com	stylesheet	255 B	18 ms	
kr0gr3orsm5xbeypbwzr1y1qiug.css	ir.ebaystatic.com	stylesheet	255 B	21 ms	
fxxj3ttftm5ltcqnto1o4baovyl.png	ir.ebaystatic.com	png	242 B	34 ms	
3ogruva0ba54dpzibdsnbwf0vul.css	ir.ebaystatic.com	stylesheet	255 B	22 ms	
CA_MERC_DD_2ndBirthday_0622_Q215_150x30.png	ir.ebaystatic.com	png	334 B	34 ms	
imgLoading_30x30.gif	ir.ebaystatic.com	gif	334 B	36 ms	
$_12.JPG	i.ebayimg.com	jpeg	242 B	64 ms	
s.gif	ir.ebaystatic.com	gif	334 B	35 ms	
$_14.JPG	i.ebayimg.com	jpeg	242 B	66 ms	

Figure 1.2: Latency of sub-requests issued for *ebay.com* webpage displaying an item.

This thesis considers 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 be 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 thesis.

The 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 centre, 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 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.

The implementation of EdgeX has been evaluated with the RUBiS benchmark [21], an *eBay* like application, on the SAVI Testbed [9]. The evaluation shows that up to 70% of the web operations can be executed at the edge sites. By handling requests at edge sites, EdgeX can achieve on an average 4x latency benefit for edgeable operations, on the SAVI Testbed. In general, this improvement will depend on the latency between the edge sites and the core site. Even in a scenario where almost half of the items are equally accessed by users of multiple regions (i.e. there exist no geo-locality in the data), EdgeX efficiently identifies the best site for each replica and maintains the latency benefit of 3.5x.

The main contributions of this work are:

1. It defines a two-tier design for web applications. This design allows web operations

to be executed close to end users, at the nearby edge sites.

2. It presents a novel two-part approach to manage replication of data between the core and edge sites. This approach consists of:
   (a) a static analysis tool that takes required information from the application developer and determines which application data can potentially be replicated at edge sites.
   (b) a runtime replica manager that determines which data should actually be replicated, and where such replicas should actually be placed.
3. It presents an evaluation of EdgeX's implementation, it's static analysis tool, best case benefits and performance with respect to geo-locality of the data.

The remainder of the thesis is organized as follows: Chapter 2 defines the basic terminology relevant to EdgeX and gives an implementation overview. Chapter 3 provides an illustration of application execution in EdgeX. Chapter 4 explains the various components of the runtime system of EdgeX. Chapter 5 explains the static analysis tool, and how EdgeX decides which data partitions are edgeable. Chapter 6 presents the system evaluation. Chapter 7 provides some of the closely related works that try to achieve the similar goals or solve some of the sub-problems, and chapter 8 concludes the thesis.

Chapter 2

System Overview

This chapter provides an overview of EdgeX's approach to utilize the two-tier infrastructure to perform web operations close to the end user. For better understanding of the approach, it is important to first know the characteristics of the infrastructure on which the web application is executed, and some assumptions and terminology about the data and the application itself.

2.1 Preliminaries

Before moving to the approach, this section will provide details about the infrastructure, the application and the data.

2.1.1 Infrastructure

The infrastructure consists of two tiers. The first tier consists of a centrally located *core* site, which is similar to present-day data centres. The second tier consists of *edge* servers or sites. Each edge site connects directly to the core site. The system has one core site and multiple edge sites. Core sites are similar to present day data centres. They store application code and a complete replica of the database of the hosted application.

Edge sites are similar to edge sites in present day CDNs. There are multiple edge sites located close to the end users. It is assumed that edges can communicate with the core site, but not with the other edge sites. The core site is linked to multiple edge sites, each

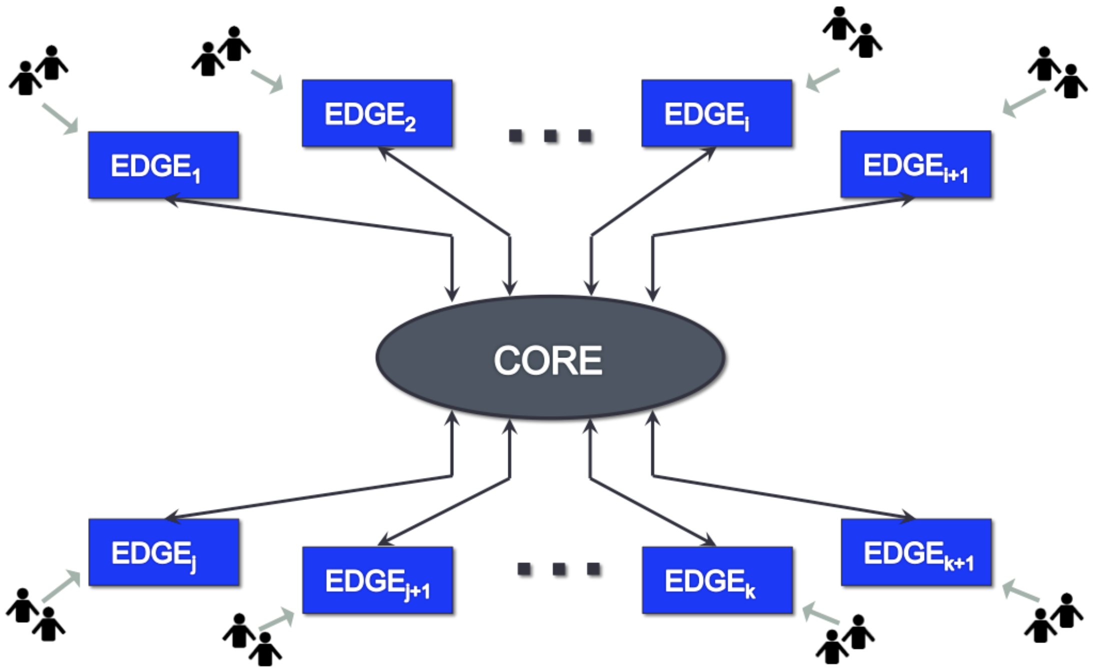


Figure 2.1: System architecture, core site with four edge sites.

representing a distinct geographical region. In other words, edges and core sites form a star network as shown in Fig 2.1. Each edge site holds application code and a partial replica of the application database. In particular, each edge contains a data partition relevant to its own geographical location. For example, in an eBay-like application hosted on a two-tier infrastructure the Toronto region edge might store data for users who buy and sell items near Toronto. Edge servers are capable of hosting the database and answering read-write database queries. A web request issued by the user is received by the nearest edge site and then it is forwarded to the core site if necessary.

2.1.2 Application model:

EdgeX assumes that the application has an Online Transaction Processing (OLTP) style workload. Applications like *amazon* [2], *eBay* [4] or *Groupon* [6] are potential use cases for EdgeX. For this thesis, *RUBiS* [21] (an eBay like auction application) will be used as an example application. The web operations consist of predefined query templates which accept parameters at runtime. Table 2.1 shows some of the query templates found in the RUBiS application. Each operation accesses data from one or more database tables. As the query templates for each operation are fixed, therefore, it is known in advance which tables are required for each operation, and the operation parameters determine which tuples in those tables are required by each operation. The clients using the application are assumed to be spread across multiple geographic locations. Clients across different regions use different parts of the database. This makes data access patterns non-uniform, and induces geo-locality in the database. In other words, some parts of the database are more frequently accessed in a specific region as compared to other regions.

Operation	Parameter	Query template
ViewBidHistory	\$itemId	`SELECT * FROM bids WHERE` *item_id* = `$itemId`
SearchByCategory	\$categoryId	`SELECT * FROM items WHERE` *category_id* = `$categoryId`
ViewMaxBid	\$itemId	`SELECT MAX(bid) FROM bids WHERE` *item_id* = `$itemId`

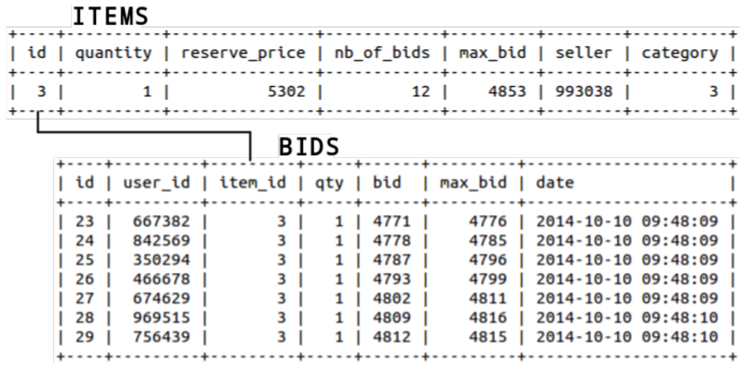
Table 2.1: Predefined query templates for eBay web operations

2.1.3 Data model

EdgeX assumes that the application uses a relational database whose schema can be partitioned into hierarchically organized *clusters* of tables, much as in Google's Spanner [24]. 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 2.2 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's tuple with primary key x, and tuples from all the other tables in the cluster with foreign key value x. For example, the cluster shown in Figure 2.2 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.



```
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 2.2: A partition of *ITEMS* cluster.

Rather, it assumes that the clusters are given. Clusters can be defined by application developers, as in Spanner, or chosen automatically [45].

EdgeX uses partition-level mastership, similar to record-level mastership in PNUTS [23], 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.

2.2 EdgeX Approach

This section provides an overview of the EdgeX's approach to executing the application's web operations. To execute web operations at edge sites, EdgeX has to take two important decisions for a given application:

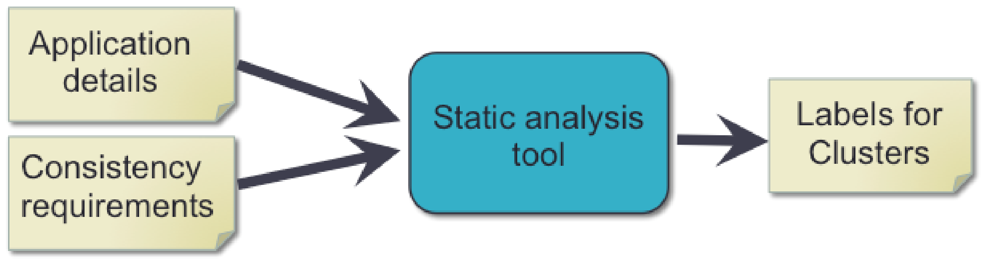
1. Which data *clusters* of the application can be replicated *safely* at the edge sites,
2. When and to exactly which edge sites should the data *partitions* be replicated.

The first decision is made statically, because it depends on the predefined query templates of the web operations and the consistency requirements, both of which stay constant during the application runtime. This decision is made at the cluster level, i.e., a placement policy for each cluster is defined. The policy indicates whether a cluster can safely have partition replicas (including master replica) at edge sites. This is possible if EdgeX can guarantee that each application operation will have at least one execution site at runtime, An operation's execution site must have sufficiently fresh copies of *all* partitions required by the operation.

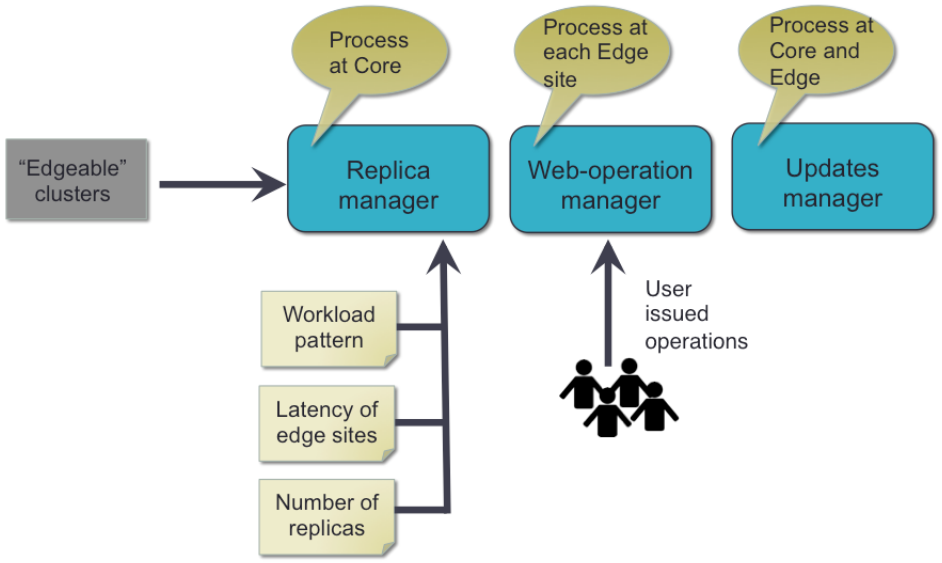
The second decision is made at regular intervals at application runtime. To make replica placement decisions EdgeX monitors the application workload and places replicas according to workload patterns, such that the maximum number of web operations will be able to execute at the edge sites.

The implementation of EdgeX is divided into two modules, each performing one of the tasks mentioned above. These modules are outlined in figure 2.3. The first module is the *static analysis* tool and is responsible for the first decision-making task, and the second module is the *runtime system*, responsible for the runtime decisions. The static analysis tool takes as input database access information and consistency requirements for each type of application operation. This information must be provided by the application developer. Given this input for all operation types, the primary task of the static analyzer is to choose a run time replication policy for each cluster. EdgeX allows two possible replication policies. The first allows the master replica to be placed either at an edge site or the core site at runtime, depending upon the workload requirements. This policy is called *Edge/Dynamic*. The second policy is called *Core/Static*. It requires that the cluster's primary partition remains at the core, and that secondary replicas be placed at all edge sites.

The second module, the runtime system, is responsible for a collection of multiple tasks that EdgeX performs at application runtime. The first task is to manage the replicas. The *replica manager* is responsible for monitoring the workload patterns at regular intervals, and for making decisions to reconfigure the replica placement if required. The replica manager tries to place the replicas at sites such that the maximum number of web operations can execute with low latency. It basically solves an optimization problem of placing replicas with above goal, given the demand for each partition at each edge, the network latency between edge sites and the core, and maximum number of replicas allowed per partition in the system. The second runtime task is to manage web operations' execution. As the replica placement changes at runtime, the execution site of web operations changes accordingly. For example, if a replica has been moved to a certain edge, the corresponding operations incoming at that edge can be executed locally, or if a replica has been deleted from certain edge, the corresponding operations need to be forwarded to the site hosting the required replica. EdgeX maintains metadata about which replica is present at each edge, and uses it to decide where an operation should execute. The third and last task is to propagate the updates between master replica and secondary replicas of each partition. The updates are propagated asynchronously in the background. Updates are propagated at the partition level, i.e., there exists a broadcast channel for each partition in the system. The master replica broadcasts updates on this channel, and all the secondary replica sites



(a) Static Analysis



(b) RuntimeSystem

Figure 2.3: System overview: two components of EdgeX.

subscribe to this channel to receive the update. As there exists a single master, updates at all the replicas are applied in the same order as they are applied at the master site.

Chapter 3

EdgeX example

This chapter's main focus is to illustrate how a web operation is handled and routed in EdgeX's two-tier infrastructure. To illustrate this, the chapter will use the RUBiS benchmark application.

EdgeX allows data partition replicas to exist at both edge and core sites, depending upon the workload. Therefore, one of the main tasks at application runtime is to execute each web operation at a site where all of the required data replicas exists. To do this, EdgeX uses the information that comes with the web operation at the runtime. Figure 3.1 shows how a web operation is handled when it arrives at the EdgeX edge site closest to the user who submits the operation. Each of these steps is described in detail in the rest of the chapter.

3.1 Edgeable operations?

In EdgeX's implementation, certain operations are edgeable, that is, they can execute at an edge site, whereas, some operations are bound to always run at the core site. When an operation is received at an edge site, the first step is to determine whether the operation should be directly shipped to the core site. An operation must be shipped to the core site for various reasons, one being that it requires access to multiple partitions and there is no guarantee that all those partitions will co-exist at a single edge site. For example, an operation like *SearchByCategory* , shown in Figure 3.4, might require access to the complete items table to search items belonging to a particular category, and each edge site stores only a part of the table. To decide whether an operation should be shipped to the core, EdgeX uses the metadata *NSMap* which stores a mapping indicating, for each web operation type, whether request of that type should be sent to the core for execution, or should be considered for execution at the edge. The information present in this map is statically generated by the *Static Analysis* tool of EdgeX, described in chapter 5. Figure 3.1 shows this first step in operation execution. Operation name and its parameters are taken as input, and the *NSMap* is used to decide if the operation should directly be sent to the core or not.

3.2 Identifying the partition

The next step is to identify which partition is required by the edgeable operation. To execute an operation at the right site, it is important to know the data requirements (exact partitions required) of each instance of the web operation. Each web operation consists of parameterized SQL query templates. Each template accesses data from certain database clusters. Information about which operation accesses which clusters is provided as input to EdgeX. The parameter passed with each operation *sometimes* defines the exact partition required by that operation. If the parameter of the operation is same as the attribute identifying the partition, then the exact partition required is known. On the other hand, if the parameter is not same as the partition-id attribute, the exact partitions required by the operation is not certain. Therefore, in the former case, by knowing the operation name and the parameter value the exact partition required at runtime for a given operation is known.

For example, in the case of RUBiS application, there is an `items` cluster consisting of *ITEMS* and *BIDS* tables (see Figure 3.3). A partition of the `items` cluster is identified by the *item_id*. Figure 3.4 shows two web operations from RUBiS application. Both of the operations are parameterized and encapsulate a query template that accepts the value of the parameter and returns the results. The first operation just views bid history for a given item and hence only accesses a single item's partition from *items* cluster, identified by the *item_id* value passed with the operation. The *NPPMap* shown in Figure 3.1 is used to identify the cluster, the partition type, and whether a primary or secondary replica of it is required for a given operation instance. For the *ViewBidHistory* operation shown in Figure 3.4, the *NPPMap* would indicate that operation requires access to either a primary or secondary replica of a partition of `items` cluster, identified by the value of the operation's *item_id* parameter. The information in this map is also statically populated by the *Static Analysis* tool.

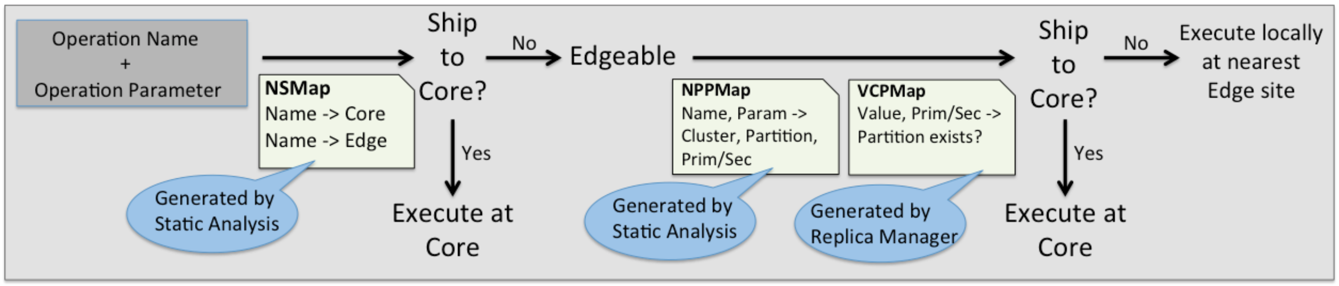

Operation Name
+
Operation Parameter
NSMap
Name -> Core
Name -> Edge
Generated by Static Analysis
Ship to Core?
No
Yes
Execute at Core
Edgeable
NPPMap
Name, Param -> Cluster, Partition, Prim/Sec
Generated by Static Analysis
VCPMap
Value, Prim/Sec -> Partition exists?
Generated by Replica Manager
Ship to Core?
No
Yes
Execute at Core
Execute locally at nearest Edge site

Figure 3.1: Complete flowchart showing the steps involved in operation execution when the operation arrives at an EdgeX edge site

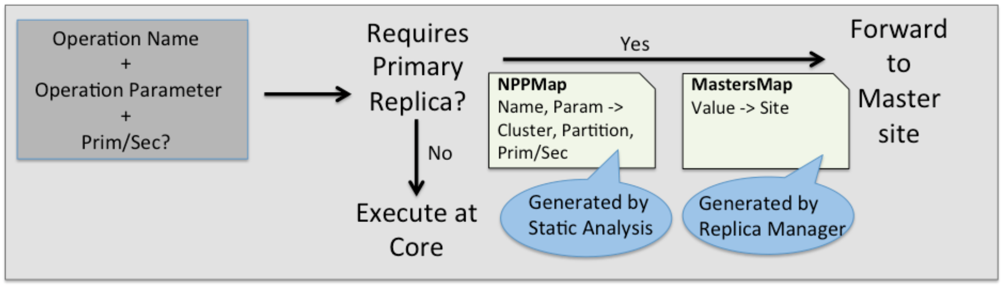
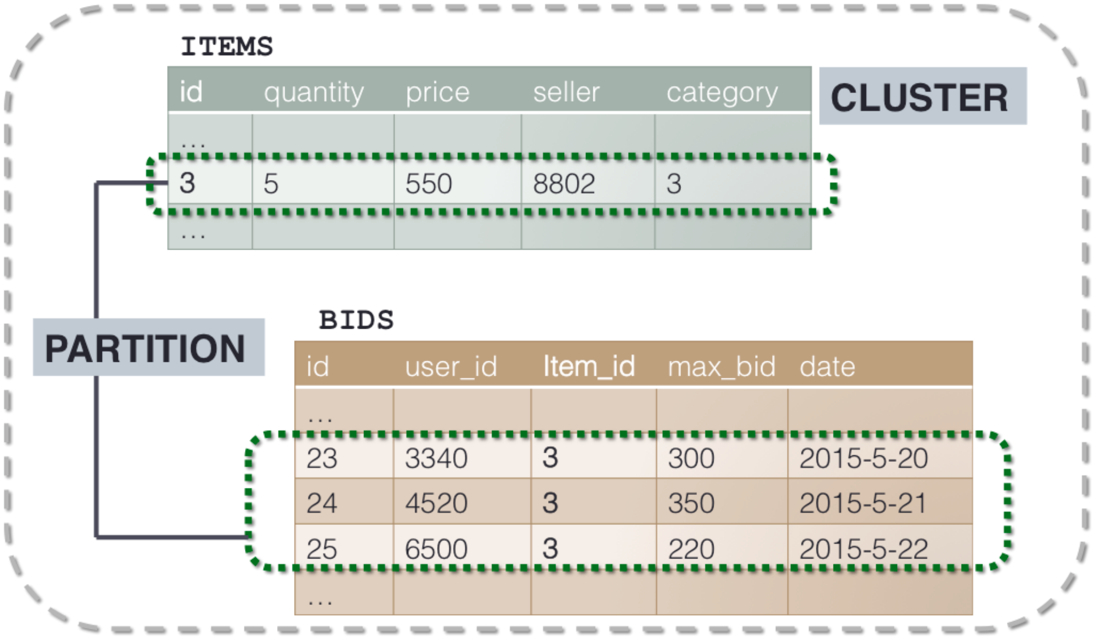

Operation Name
+
Operation Parameter
+
Prim/Sec?
Requires Primary Replica?
Yes
No
Execute at Core
NPPMap
Name, Param -> Cluster, Partition, Prim/Sec
Generated by Static Analysis
MastersMap
Value -> Site
Generated by Replica Manager
Forward to Master site

Figure 3.2: Complete flowchart showing the steps involved in operation execution when the operation arrives at the core site.



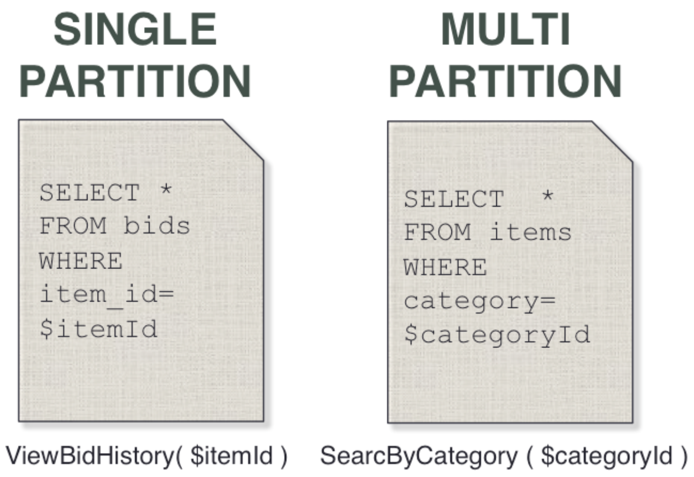
ITEMS

id	quantity	price	seller	category
…				
3	5	550	8802	3
…				

BIDS

id	user_id	Item_id	max_bid	date
…				
23	3340	3	300	2015-5-20
24	4520	3	350	2015-5-21
25	6500	3	220	2015-5-22
…				

Figure 3.3: Items cluster with partitions.



```
SELECT *
FROM bids
WHERE
item_id=
$itemId
```

```
SELECT  *
FROM items
WHERE
category=
$categoryId
```

Figure 3.4: Two of the RUBiS web operations.

3.3 Locating the partition

Once the partition required for a given web operation instance has been identified, the next thing is to determine whether the required partition is available at the edge site or not. If not, the operation is forwarded to the core site and then core site decides whether the operation should execute at the core or whether it needs to be sent to the remote edge site. To decide this, EdgeX maintains another map, the *VCPMap*, which records the primary and secondary replicas that are currently present at the site. When the operation is received at the edge site, EdgeX first identifies the required partition replica, and then checks the *VCPMap* to determine its availability. To consult *VCPMap*, the value of the parameter is obtained from the operation and the information — whether a primary or secondary replica is required — is obtained from the *NPPMap*. Given the value and type of replica required the *VCPMap* determines whether the required replica exists at the site or not. Contents of *VCPMap* are maintained by the replica manager. Every time a replica is moved to or from an edge site, the replica manager updates the *VCPMap*. Figure 3.1 shows the format of VCPMap metadata. Identifying the required partition and checking its availability using the VCPMap together decide whether this edgeable operation can actually execute at this edge site or needs to be forwarded to the core site.

The maintenance of metadata is crucial because EdgeX picks an execution site using it. If the state of metadata is outdated then it might result in operation failure at the site. To ensure that the metadata does not return false positives, movement of replicas and updating the corresponding metadata is done atomically. For example, if EdgeX decides to place a secondary replica at Edge site i, then placing a replica and appending this new replica to the metadata of site i is done atomically. Any operations received during the replica movement process are forwarded to the core.

Figure 3.2 shows how requests that are shipped to the core are handled by EdgeX. The core has a replica of all the data partitions, but they might not necessarily be the primary replicas. Thus, the core site also has to maintain metadata about primary replicas which exist at the edge sites. This metadata is consulted when an operation requiring a primary replica is forwarded to the core site, see *MastersMap* in Figure 3.2. The core site identifies the required partition, and consults the metadata to find out where the primary replica is located, and then redirects the operation to that site. This scenario occurs when the primary replica exists at an edge site other than the one that first receiving the operation. EdgeX runtime system tries to place replicas so that this scenario happens rarely.

Chapter 4

Runtime System

The previous chapter explained how EdgeX performs the main task of executing the application operations on a two-tier infrastructure, when replicas are placed according to the workload. However, other details that were left unexplained are how replicas are placed according to the workload, how are the metadata (e.g. *VCPMap*) maintained, and how are updates propagated between replicas. This chapter first presents an overview of the rationale for the EdgeX system design and a discussion of major design decisions. It then gives a complete overview of EdgeX's runtime system which consists of three major parts: replica management, operation execution and updates management.

4.1 EdgeX Design Rationale

The main objective of EdgeX is to reduce the latency of web operations by executing them at the nearest edge site. To execute the operations at edge sites, there are several design choices that must be made:

1. Should replicas be synchronized lazily or eagerly?
2. What consistency guarantees should EdgeX provide.
3. If a single edge site does not have all the required data, should operation execution be distributed across multiple sites?

These design questions are discussed in detail in the remainder of this subsection.

4.1.1 Synchronous vs. asynchronous replication

While storing replicas at edge sites, EdgeX wants to achieve more than just caching the data close to the user. EdgeX wants to utilize the edge sites to handle updates as well as queries. This makes EdgeX more useful than the existing content delivery networks, which only cater to read-only data. However, performing updates at edge sites is beneficial only if the latency is lower compared to performing them at the core site.

To achieve fast update operations at edge sites, EdgeX sacrifices strong consistency, it uses an asynchronous mechanism to propagate updates. To have simple and low latency update transactions it also prefers a single master model over a multiple master model where conflict resolution can be difficult. Therefore, in the EdgeX system, each partition has a single master replica and multiple slave replicas. The master replica can exist at an edge site or at the core site, depending upon the workload pattern. An edge site can perform update operations on a partition only if it is the master site of that partition. An update transaction is considered committed if the master site has locally committed the data. Once the transaction is locally committed, the updates are propagated asynchronously to the remaining sites holding the replica. EdgeX monitors the workload pattern at runtime, and can adjust the location of the master replica to suit the workload.

4.1.2 Application specific consistency

The choice of asynchronous master-slave replication in EdgeX does not allow for strong consistency guarantees. However, there may be certain web operations in the application which always require fresh data. For example, in an eBay like application, the operation to get the maximum bid on an item should never return a stale bid. For EdgeX's approach to be more broadly applicable, it has to be flexible enough to handle such web operations.

EdgeX satisfies consistency requirements at the web operation level, rather than guaranteeing consistency at the database level. For this, it requires the consistency requirements of each operation. EdgeX assumes that the application developer is the best judge of operations that are critical to the system and require fresh data. Therefore it takes this information as input from the developer. An application developer might always prefer fresh data for all operations, and EdgeX can guarantee that. However, EdgeX may not be able to provide any performance gains in that case. Therefore, it is always in the best interest of the application developer to demand fresh data for as few operations as possible. Operations which require fresh data are always executed at the site of the master replica. The master replica site can either be the closest edge site, the core site, or a remote edge site. In the case of closest edge site, the latency benefits are greatest, whereas if the operation is executed at the core site, there are no benefits. Executing the operation at a remote edge is the most expensive, because the operation makes a round trip between two edge sites via the core site. EdgeX tries to place master replicas such that as many operations requiring fresh data as possible are executed at the nearest edge site.

4.1.3 Distributed transaction vs. single-site transaction

At the application runtime, there may be cases in which the edge site does not contain all the data required for an incoming web operation. This usually happens when the edge site receives operations requiring new partitions (partitions whose replica is not yet present at this edge). Two possible choices, in that case, are: 1) perform a distributed transaction over the sites containing the required partitions, 2) distribute data between sites such that there is always at least one site that contains all the required data partitions, and execute the operation completely at that single site. Performing a distributed transaction is slow and costly, because it can involve multi-phase locking and message passing between distant sites. Therefore, to have fast transactions, EdgeX ensures that each operation is completely executed at a single site. This site can be an edge site if it contains all the required partitions, or a core site which by default contains a complete replica of the data. Single-site execution is guaranteed by EdgeX's static analysis of the data and consistency requirements of the application's operation types. The results of this analysis are used to constrain data placement, so that single site execution can be guaranteed.

4.2 Runtime System Components

The run-time system of EdgeX has several responsibilities. First, it decides the exact site for each primary and secondary replica of each partition. Second, based on the location of the replicas it decides the exact execution site for each requested operation. The location of replicas and the execution site of operations changes with changing workload pattern. Third, it propagates updates between primary and secondary replicas.

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

4.2.1 Replica Manager

For global web applications, the data access patterns are non-uniform across various geographic locations. Work by Agarwal et. al. [12] shows certain data access patterns for web applications and leverages them to automatically place data in geo-distributed cloud services. For example, in data driven websites like *eBay*, an item may be expected to receive more requests from the local geographic location where it is being sold, than from other, distant geographic locations. This means that certain partitions are more frequently accessed at certain geographic locations. If the edge site representing a certain geographic region is able to locally handle the requests originating in that region then those requests will have lower response latency. Therefore, the replica manager's main task is to identify the data access patterns, and place the replicas accordingly. It needs to know which partitions are more frequently accessed at which edge sites, and propose an optimal site for replicas such that web operation have lower response latency.

Replica manager is a central service that runs in the background and periodically takes note of the workload observed at each edge site. Observing the workload periodically enables the replica manager to account for changing workload patterns. For example, in an *eBay*-like application, if the buyer of an item moves from one geographical region to another, chances are high that the demand of that item will increase in the new region and decrease in the old region. Therefore, it is beneficial if this change in workload is captured and the data partition of that item is moved to the new region.

Another situation in which a replica placement becomes important is when a partition is almost equally demanded by two or more edge site's regions. Only a single primary copy of each partition can exist. The choice of primary site becomes complicated, as it depends on the latency between the core and each edge site. The site which is farthest from core should hold the replica, because the latency benefit will be higher. However, if all edge sites have almost equal latency from core then it might be beneficial to leave the replica at the core site.

It is clear from the above two workload scenarios that the decision of replica placement depends on two parameters: the demand of each partition from each edge site, and on the latency between the core site and each edge site. To decide on an optimal placement of replicas, this information is taken as input. To get the latency, replica manager pings each edge site periodically. To get the demand of each partition replica at each site, it collects operation request logs from each site and uses them to identify how many web operations access primary and secondary replica of each partition. The partition required by each operation is identified from its parameters. In addition to this input, replica manager has a parameter to control the number of replicas of each partition. It is used to control the cost of maintaining new replicas.

The objective of the replica manager is, given the above inputs, to assign a site for each partition replica such that overall latency of requests accessing that partition across the sites is minimized. This problem is modelled as a constrained optimization problem. The problem has two boolean variables for each edge. One of the boolean variables indicates whether an edge site should get the primary replica, and the other variable is to indicate whether the site should have a secondary replica. The solution to the problem is an assignment to these boolean variables. The objective function is to minimize the overall latency across the system for the target partition.

The definitions and the formulation of the problem is presented below:

For each partition P in the system -

- N: Number of edge sites in the system.
- f: Replication factor - Maximum number of secondary replicas allowed for partition P. $f \leq N$.
- l_i: Latency between Core and Edge site i. $i \in \{1, ..., N\}$.
- s_i: Secondary replica demand at edge i. Basically, number of times secondary replica has been accessed at edge i since the last check.
- p_i: Primary replica demand at edge i. Number of times primary replica has been accessed, at edge i, since last check.
- x_i: Boolean variable. If 1, place a secondary replica at edge site i, otherwise, do nothing.
- y_i: Boolean variable. If 1, place a primary replica at edge site i, otherwise, do nothing.

Optimization problem:

Total latency for the operations that are fine with stale data.

$$L_{secondary} = \sum_{i=1}^{N} s_i \times l_i \times (1 - x_i - y_i) \tag{4.1}$$

Total latency for the operations that require fresh data.

$$L_{primary} = \sum_{i=1}^{N} (p_i \times l_i \times (1 - y_i) + (\sum_{j=1, j \neq i}^{N} p_j) \times l_i \times y_i) \tag{4.2}$$

Constraint equations:

Number of secondary replicas should be limited to f.

$$\sum_{i=1}^{N} x_i \leq f \tag{4.3}$$

Number of primary replicas should be 1.

$$\sum_{i=1}^{N} y_i = 1 \tag{4.4}$$

Same site cannot have both primary and secondary replicas.

$$x_i + y_i \leq 2 \tag{4.5}$$

Boolean variable.

$$x_i \in 0, 1 \forall i \in \{1, 2, ...N\} \tag{4.6}$$

Boolean variable.

$$y_i \in 0, 1 \forall i \in \{1, 2, ...N\} \tag{4.7}$$

Objective function:

Minimize the total latency.

$$Minimize \quad L_{total} = L_{secondary} + L_{primary} \tag{4.8}$$

Explanation: The objective function assume that a request's latency is zero if it can be executed at the site where it arrives. Otherwise, the request's latency is determined by the input edge-to-core latency (l_i). Equation 4.1 represents the total latency of all the operations that access secondary replica of partition P. By default it is assumed that the data exists at the core site, therefore latency to execute the requests at this site is

$s_i \times l_i$. When the replica of the partition is moved to edge i, i.e. either x_i or y_i is set to 1, the latency cost paid for transferring data from core to edge becomes zero. For equation 4.2, there are two costs. The first is paid to get fresh data from core site if it does not exist at the edge site, i.e. $p_i \times l_i$. In addition, if some edge site i holds a primary replica then any other site demanding fresh data has to forward the request to this site, hence, $(\sum_{j=1, j\neq i}^{N} p_j) \times l_i$.

The values of variables x_i and y_i defines which edge site i should receive the secondary and primary replicas respectively. Note that there are no variables that represent the placement of partition replicas at the core site because the core always at least has a secondary replica and if no edge site has the primary replica the core site has it. This optimization is performed separately, and independently for each partition accessed in that time interval. The new values of the variables are compared with their previous values to identify which data partitions need to be moved. If the value of variable changes, it means that the workload pattern has changed and location of certain replicas should be changed. With the change in location of replicas the *VCPMap* is updated, that is how each edge site knows about the current set of replicas it holds.

4.2.2 Operation Request Handling

Each web operation is first received at the closest edge site. It is the edge site's responsibility to decide which web operations should be forwarded to the core and which can be handled locally. Operations which are non-edgeable are directly forwarded to the core site; the static analysis tool determines which operations are non-edgeable. Operations which are edgeable, but for which the edge site does not have the required data, are also forwarded to the core site. Figures 3.1 and 3.2 give an overview of web request handling in EdgeX.

EdgeX forwards the request at the Apache web server level, i.e. requests which are forwarded do not execute any application code at the edge site. This minimizes the overhead of request forwarding. In the EdgeX prototype, Apache's *mod_rewrite* module is used to forward web requests. Apache *mod_rewrite* performs rule-based rewriting and forwarding of the web operation URLs. It uses a set of rules to decide which URLs need to be rewritten, and then rewrites them with the hostname of the site which contains the required partition. The rules for non-edgeable operations are static, and are part of the *NSMap* metadata shown in Figure 3.1. Rules for edgeable operations are dynamic, because the location of replicas can change with time. For edgeable operations, the *mod_rewrite* module consults the *NPPMap* and *VCPMap*, stored in form of *.dbm* files at each edge site, to decide which operation should be forwarded.

Metadata in *.dbm* files is in the form of key-value pairs as shown in Figure 3.1. For *VCPMap* the key is the ID of partitions present at that edge site, and the value is 'localhost', because operations requiring those partitions should be executed locally. The URL is rewritten with this value as the new hostname. If certain partition ID does not exist in the *.dbm* file, it means that edge should forward the corresponding operations to the core site. There exist two *.dbm* files for each Edge/Dynamic cluster, one for primary replicas and one for secondary replicas. Entries in the *.dbm* file are updated when the placement of replicas is changed, which is determined periodically by the replica manager. If an edge receives a new replica then it adds an entry for that partition ID with 'localhost' as the value. In case a replica is deleted from an edge site, the entry for that partition is deleted too.

Web operations which require a primary replica are always forwarded to the site containing the primary replica. This forwarding is done in two steps: First, if the edge site receiving the operation is not the primary site for the required partition, it forwards the operation to the core site. Second, the core knows which edge site contains the primary replica of the required partition, and forwards it to that edge site. The core site also maintains metadata about where the operations should be forwarded, in the *MastersMap* shown in Figure 3.2. It uses the Apache *mod_rewrite* module and *.dbm* files to implement operation forwarding.

4.2.3 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. The update propagation mechanism ensures that all updates to a partition are serialized by the primary copy of that partition, and then propagated lazily and in order to any secondary copies.

A single database table can have different masters,one for each partition. Popular techniques for master-slave replication for complete tables are not directly useful for EdgeX. Updates for individual partition need to be channeled to specific sites. Each site will be accepting updates for all the secondary replica it hosts and sending updates for each primary replica it hosts.

The EdgeX prototype uses open source replication software called *SymmetricDS* [10]. SymmetricDS uses a publish-subscribe mechanism to propagate database updates lazily between sites. It defines a publication channel for each 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 through the appropriate channels. Internally, symmetricDS uses triggers to capture changes, and categorizes those changes into different channels. For example, if the quantity of an *item* partition with partition ID 5 is updated, the changes will be captured by the channel for the partition with ID 5 and then pushed.

Chapter 5

Static Analysis

This chapter provides a detailed description of EdgeX's *static analysis* tool, which is responsible for determining which partitions are edgeable. It addresses the following concerns: Why is the static analysis of the web application important? What are the challenges that EdgeX needs to overcome? What approach does EdgeX use to solve these challenges? The chapter concludes by demonstrating the effectiveness and benefits of EdgeX's static analysis tool with the help of an OLTP style benchmark.

5.1 Data access in web applications

A typical user-data-driven web application requires a complete database replica, stored at the data centre, to execute the web queries. This is because at runtime it is not certain which tuples of the database will be needed to generate a query response. A complete replica of the database stored at a single site ensures the execution of any possible valid query. However, if the application has a fixed set of pre-defined web operations, then a complete replica of database is not required. This is because pre-defined web operations ensure that required tuples are confined within a certain set of tables for each operation. For example, a web operation that is used to view details of an item on a bidding website always uses the *items* table and not the *users* table. The exact tuples needed for each operation depend on the value of the parameters of the web operation, which are known only at runtime. Using the definition of a *cluster* from the previous section, EdgeX can assume that each web operation requires tuples from a fixed set of clusters. If the set of required clusters for each operation is confined, then it is advantageous for EdgeX, because the execution site of each operation now only depends on the placement of the required clusters. This can potentially allow EdgeX to place certain clusters at the edge sites and execute the corresponding web operations at those edge sites. The application developer is expected to provide the information about *which operations require which clusters*. The tool uses this information in combination with two other constraints, described later, to decide which clusters can have primary replicas at edge sites and which web operation can execute at the edge sites.

In addition to knowing which clusters are accessed by each operation, it is also important to know the exact tuples which will be required by each operation. It is important because with this information EdgeX can place those tuples at the edge sites and allow specific operations to execute at these edge sites. For certain types of web operations, if the operation parameters are known, then by using the parameter value EdgeX can decide which tuples will be required. For example, a web operation that shows details of an item on a bidding website accepts item Id value in the parameters and selects the tuples specific to the provided item Id only. These operations are classified as *single-partition* access operations, as they access only a defined set of tuples identified as a partition, this is more formally explained in the next section. On the contrary, for certain operations, parameter's values are not sufficient to identify the required tuples. For example, an operation that returns the maximum bid on an item has to get all its bid. Therefore for this operation, result is dependent on all the bid entries of a given item. Such operations are classified as *multi-partition* access operations, as they can access multiple set of tuples. These are also explained in next section. The information about the way (i.e. single-partition or multi-partition) each operation accesses the required cluster is provided by the software developers. This information is also a part of input provided to the static analysis tool of EdgeX.

The above inputs describe the granularity at which the data requirements of an operation can be known in advance, i.e. which clusters, and which tuples within a cluster will be accessed by an operation. The static analysis tool utilizes this information along with the consistency requirements, explained in next section, and then assigns a placement policy for the clusters and an execution policy for the web operations.

5.2 Challenges and Problem statement

EdgeX chooses at application runtime where to place primary and secondary replicas 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. However, EdgeX's placement choices are constrained in the following two ways.

- First, EdgeX must ensure that it is possible to execute each application operation at a single execution site: either at an edge site, or at the core site, because EdgeX does not perform distributed transactions. The overhead of performing a distributed transaction over multiple edge sites will dominate the latency benefits obtained by executing the operation at a nearby edge site.
- 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.

The static analysis tool's challenge is to assign placement and execution policies such that the above constraints are satisfied. These policy assignments are then used as input to EdgeX's runtime system. EdgeX's runtime system wants to know two things: First, which data partitions can it replicate at edge sites. Second, which type of web operations can it execute at edge sites.

EdgeX assumes that the application's server-side code implements a set of parameterized operation types $\{O_1, ..., O_n\}$. Handling each type of operation 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 operation. This information must be provided by the application developer. Specifically, for each operation O_i, the analysis is given:

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

We say that O_i requires *single-partition* access to a cluster C if it can be determined that each operation of type O_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 operation parameter. For example, if C contains information about users and is partitioned by *UserID*, O_i requires single-partition access to C if O_i takes a *UserID* as a parameter and only reads or writes data in that user's partition of C. Different instances of O_i may access data about different users, but each instance must be known to access data about a single user. Operation types that are not single partition are called *multi-partition.*

The consistency constraints for O_i on cluster C indicate whether O_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 O_i requires fresh access to C, then it must run where the primary copy of C is located. Otherwise, O_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 operation type that updates a cluster requires *fresh* access.

Given this input for all operation 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 run-time.

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 mentioned at the beginning of this section. To understand these constraints' effects on policy assignment, consider the following example cases:

Case 1: Suppose operation type O_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 operations of type O_i could run.

Case 2: Suppose that operation type O_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 O_i operation 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 operations of type O_i can run at the core.

Using similar reasoning, the static analysis tool 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 the 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.

5.3 Approach

The problem of determining a placement policy for clusters and execution policy for operations is modelled as a constrained optimization problem. The output of static analysis is expected to be a placement policy, either *Core/Static* or *Edge/Dynamic*, assigned for each cluster, and an execution policy, either *atCore* or *atEdge*, assigned for each web operation. This execution policy is stored as static metedata *NSMap* at each edge site, as shown in figure 3.1. In a scenario where each web operation accesses only one cluster, and no two operations access the same cluster, the task of assigning replication policies would be trivial, because the policy for each cluster will depend only on the way it is being accessed by a single operation. On the other hand, if there are multiple operations accessing different types of clusters with different consistency constraints, the task of assigning policies becomes complicated. Case 2 in section 5.2 demonstrates a sample scenario where different types of clusters are accessed by a single operation, and with different consistency constraints. The key idea of the approach in this scenario is to generate policy constraints corresponding to each consistency requirement specified by the application, and then choose an optimal policy assignment that satisfies all such constraints and allows maximum number of operations to execute at edge sites.

EdgeX implements the above approach in two steps: 1) generating policy constraints corresponding to the inputs given to the static analysis tool; 2) choosing an optimal policy assignment that satisfies all the constraints generated in the previous step.

STEP 1: The static analysis tool considers each web operation individually, along with the inputs about what clusters it accesses and what consistency guarantees are required. Then, for each operation, it transforms the inputs into possible policy assignments, each allowing the operation to execute at one site (either an edge site, or the core site), or at both, or at no site. For example, suppose that an operation type O_i requires fresh, single-partition access to cluster C_j and stale single-partition access to cluster C_k. Irrespective of the input constraints, there are total four possible policy assignments: both clusters get policy Edge/Dynamic, both clusters get policy Core/Static, C_j gets Edge/Dynamic and C_k gets Core/Static, or C_j gets Core/Static and C_k gets Edge/Dynamic. Now, when the inputs about consistency requirements and the way O_i accesses each cluster, i.e., single-partition or multi-partition access, are considered, some of these assignments become invalid, because they do not allow O_i to execute at any single site. In this example, assigning policy Edge/Dynamic to both clusters is invalid, because at runtime the system cannot ensure that partitions of both the clusters co-exist at a single site, hence there may not exist a single site where operation can execute. Therefore, a valid policy assignment is one that allows the operation to execute at some site.

To figure out which of the possible assignments are valid, the static analysis tool uses a procedure $feasible()$. This procedure takes input constraints and a policy assignment as input, and tells whether a given assignment is valid or not. If valid, it outputs whether the operation is allowed at edge sites, or the core site, or both. If invalid, it outputs $false$. This procedure uses reasoning similar to that described in the two cases in section 5.2. For now, it can be considered as an oracle capable of telling which assignment is valid and which is not. Details will be presented later in this section.

In the first step, the static analysis tool generates all possible policy assignments for each operation. Then, it uses the $feasible()$ procedure to filter out invalid policy assignments. All the remaining valid assignments for each operation are logically disjunctive in nature, because each of those assignments, if satisfied, can individually allow the operation to execute at some site. Step 1 generates disjunctive expressions of policy assignment for each operation. These expressions are separated into two categories: one, that allows the operation to execute at the edge site, another, that allows the operation to execute at the core site. The tool separates the expressions, because its goal is to find an assignment that satisfies all the constraints and also allow maximum number of operations to execute at edge sites. Therefore, the output of step 1 is two type of disjunctive expressions for each operation. These are fed as an input to the next step.

STEP 2: The best policy assignment for the clusters is one that satisfies all the policy constraints and also allows the maximum number of operations to execute at the edge sites. The problem of finding the best policy assignment is modelled as a constrained optimization problem. The input to this problem consists of the two type of expressions received from the previous step.

The problem variables, and the formal definition of expressions (in terms of problem variables) are provided in this section. Each cluster has two variables, representing two possible replication policies. Thus, a cluster C_i has variable x_{i_ED} for *Edge/Dynamic* policy and x_{i_CS} for *Core/Static* policy. These variables can take values of either 0, meaning policy is not valid, or 1, meaning policy is valid. The disjunctive expression consists of conjuncts formed by these variables, making the expression in disjunctive normal form (DNF). For each conjunct, there is exactly one variable, representing the corresponding policy for each cluster it requires. $edge_j(X)$ represents the DNF form of expression that allows the operation to execute at the edge sites, where j is the operation and X denotes a possible value assignment to the cluster variables, i.e., a set of decision variables x_{i_ED} and x_{i_CS}. Similarly, expression $core_j(X)$ represents the DNF form of expression that allows the operation to execute at the core site.

To form an optimization problem from the given expressions, the static analysis tool replaces logical AND ($\wedge$) symbols of the conjuncts with *product* ($\times$) operation and logical OR ($\vee$) operation of the disjunctive expression with *sum* (+)operator. Next, it requires an objective function and constraints. The first constraint is that each operation must have an execution site, meaning that the value of either $edge_j(X)$ or $core_j(X)$ should be greater than zero. Next, each cluster must have exactly one assigned policy; therefore, exactly one of x_{i_ED} or x_{i_CS} should have value 1. The objective function is to maximize the number of operations that can run at edge sites; consequently, the sum of $edge_j(X)$ for all j should be maximum.

More formally, the problem is as follows: For each of the M clusters C_i, there are two variables x_{i_ED} and x_{i_CS}. For each of the N operations O_j, there are two expressions, $edge_j(X)$ for O_j executing at an edge site, and $core_j(X)$ for O_j executing at the core site.

Constraint equations :

These constraint ensures that at least one execution site exist for the operation.

$$edge_j(X) + core_j(X) > 0, \quad j \in \{1, ...N\} \tag{5.1}$$

These constraint ensures that only one of the boolean variable is set to 1.

$$x_{i\_ED} + x_{i\_CS} = 1, \quad i \in \{1, ...M\} \tag{5.2}$$

Domain of the two variables.

$$x_{i\_ED} \in \{0, 1\}, \quad i \in \{1, ...M\} \tag{5.3}$$

$$x_{i\_CS} \in \{0, 1\}, \quad i \in \{1, ...M\} \tag{5.4}$$

Objective function:

$$Maximize \quad f(X) = \sum_{j=1}^{N} edge_j(X) \tag{5.5}$$

The number of variables in the above optimization problem is $2M$, i.e. in order of number of clusters in the application. For the current benchmark application number of clusters is only 3 so EdgeX uses a brute-force approach to check all possible variable assignments. In practice EdgeX can use an off-the-shelf optimization solver to calculate the best variable assignment.

The summary of the whole approach is to first exhaustively enumerate all possible assignments, then pick the ones that are valid, then out of those pick one for which maximum number of $edge_j(X)$ expressions are *true*. Algorithm 1 briefly outlines this approach. The input is list of operations along with the clusters it accesses. Output is a policy for each cluster. The outer loop iterates through all the operations. For each operation procedure *enumAllAssig()* generates all possible policy assignments. Procedure *applyRules()* generates the conjuncts for a given operation using the rules defined below and then these conjuncts are added to the $edge_j(X)$ and $core_j(X)$ expressions based on the decision of $feasible()$ procedure. It then adds the above expressions to the constraints and the objective function. Second inner loop iterates over each cluster accessed by the operation and generates the constraint expression of 5.2 and adds it to the existing list of constraints if its not already added. At the end the $optimize()$ procedure outputs the policy assignment that satisfies all the constraints and maximizes the number of operations that can run at edge sites.

Once the policy for each cluster is assigned after optimization, the next task is to assign an execution policy for each operation. Assigning this depends on which of the $edge_j(X)$ or $core_j(X)$ expressions are true. The operation is considered to execute at an edge site if $edge_j(X)$ is true, and sent to the core otherwise. If both the expressions are true then it is always beneficial to consider it for execution at an edge site.

feasible(): A given policy assignment is called *valid* if it allows the operation to execute at some site, either edge site, or core site, or both. The task of this procedure is to identify if a given policy assignment is valid or not, and if it is valid then output whether it can run at edge sites, or core site, or both. To decide the validity of a given assignment, this procedure consults a set of rules. These rules consider the input constraints, and the given assignment to decide if the assignment is valid or not. The rules are based on similar reasoning as presented in Case 1 and Case 2 of section 5.2.

Rules tell if the assignment is valid and if yes what can be the execution policy for the operation. More formally the rules are defined as following :

- **Rule 1:** If an operation requires fresh, multi partition access to one or more clusters, then the assignment where all such clusters have policy Core/Static will only be valid (allowing operation to execute at the core site, i.e., $core_j(X)$ is only true). Any other assignment is invalid (both $core_j(X)$ and $edge_j(X)$ are false). This is because at runtime it is not certain that any single edge site will contain fresh replica of a complete cluster.

- **Rule 2:** If according to the assignment, more than one cluster is assigned policy Edge/Dynamic. The assignment is valid but the operation can execute only at the core site. (Only $core_j(X)$ will be true.) This is because it is not certain that at runtime multiple partitions exist together at a single site.

- **Rule 3:** If an operation requires fresh data from more than one cluster. Then, the assignment where all such clusters have policy Core/Static is only valid (allowing operation to execute at the core site, i.e., $core_j(X)$ is only true). Any other assignment is invalid (both $core_j(X)$ and $edge_j(X)$ are false). This is because only core site is guaranteed to store the master replica of multiple clusters by choosing policy Core/Static for all those clusters.

- **Rule 4:** If any of the above rules do not apply, then the assignment is valid and the operation can execute at both, edge sites and the core site (both $core_j(X)$ and $edge_j(X)$ are true).

Next section will demonstrate how the static analysis works on the RUBiS [21] benchmark. It will demonstrate how expressions are formed, and what upper-bound does it give on the maximum number of operations that can execute at edge sites during runtime.

5.4 Static Analysis on RUBiS

RUBiS [21] is an ebay like application where users can buy, sell and put bid on items on sale. It has a relational database at the backend with 7 tables. There are 17 predefined, parameterized web operations that access and write data into the 7 database tables. It is assumed that the clustering of tables, and access information about the operations is given, i.e. which operations access which clusters and with what consistency requirements.

```
Data: Set of operations with clusters they access and the inputs. $Operations : \{O_i : \{C_j : (Fresh/Stale, SinglePartition/MultiPartition), ..\}, ..\}$
Result: Value assignment for each cluster variable. $\{x_{i\_CS} = 0/1, \{x_{i\_ED} = 0/1, ..\}$
$constraints$ = null;
$objectiveFunc$ = null;
for each operation $O$ in $Operations$ do
    $A$ = enumAllAssig($O$);
    for each assignment $a$ in $A$ do
        $sites$ = feasible($a$, $O$);
        $conjunct$ = applyRules($O$);
        if "edge" in $sites$ then
            $edge_j(X)$ += $conjunct$;
        end
        if "core" in $sites$ then
            $core_j(X)$ += $conjunct$;
        end
    end
    $constraints$.add("$edge_j(X) + core_j(X) > 0$");
    $objectiveFunc$ += $edge_j(X)$;
    for each cluster $i$ in $O$ do
        $expr$ ="$x_{i\_CS} + x_{i\_ED} = 1$";
        if $expr$ not in $constraints$ then
            $constraints$.add($expr$);
        end
    end
end
$values$ = optimize($constraints$, $objectiveFunc$,"Maximize");
return $values$;
```

Algorithm 1: Static Analysis

For simple demonstration of how static analysis works, two clusters and two operations from the RUBiS benchmark are used. The results of static analysis on the complete RUBiS benchmark are shown later.

Two clusters for this example are *users* cluster, consisting of tables *Users* and *Comments*, and *items* cluster, consisting of tables *Items* and *Bids*. Operation $ViewUserInfo$ returns user information for a given user Id, it requires single partition, stale access from *users* cluster. Operation *PutComment* stores comment by a user for a given item Id and user Id, it requires single partition, stale access to two clusters *users*, and *items*. As operation $ViewUserInfo$ accesses only one cluster, there are only two possible assignments, and both these assignments are feasible and let the operation run at the core as well as at an edge site.

The expressions for $ViewUserInfo$ are:

$edge_{ViewUserInfo}(X) = x_{users\_CS} + x_{users\_ED}$

$core_{ViewUserInfo}(X) = x_{users\_CS} + x_{users\_ED}$

ED stands for policy Edge/Dynamic and CS stands for policy Core/Static. For operation *PutComment* both the clusters can get both the policies, resulting in four possible assignment. Of these, it can execute at an edge site only for three assignment either, *users* has Edge/Dynamic and *items* has Core/Static policy, or *users* has Core/Static and *items* has Edge/Dynamic policy, or both have Core/Static policy. For the fourth combination, where both clusters have Edge/Dynamic policy, the operation cannot execute at the edge site because of *Rule 2* from previous section. Whereas, to execute *PutComment* at core, all four combinations are possible.

The expressions for *PutComment* are:

$edge_{PutComment}(X) = x_{users\_ED}.x_{items\_CS} + x_{users\_CS}.x_{items\_ED} + x_{users\_CS}.x_{items\_CS}$

$core_{PutComment}(X) = x_{users\_ED}.x_{items\_CS} + x_{users\_CS}.x_{items\_ED} + x_{users\_CS}.x_{items\_CS} + x_{users\_ED}.x_{items\_ED}$

Given the above expressions, the optimization problem can be formalized using equations [5.1 - 5.5]. An optimal solution to this optimization problem is to have Core/Static policy for *users* cluster and Edge/Dynamic policy for *items* cluster, and it allows both the operations to execute at the edge sites during runtime.

Table 5.1 shows the input and result of static analysis done on the complete RUBiS application. It shows all the web operations and clusters in matrix form. The entries in the matrix shows which operations access which clusters and with what properties. The policies (abbreviated to ED and CS) assigned for the clusters are written next to them, and policies for operations are in the third column. Second column shows the percentage distribution of each type of operation in the RUBiS workload. Consistency requirements for each operation with respect to each cluster are written in the corresponding cells. Cell colours show the access fashion for each operation with respect to a given cluster. Blue cells show single partition access, and green cell shows multi partition access to the cluster. The resulting policy assignments allow upto 70% of RUBiS application operations to be handled entirely at the edge sites.

Operation	**Share**	**Policy***	**USERS (CS*)**	**ITEMS (ED*)**	**BUY NOW (CS*)**	**REGIONS (CS*)**	**CATEGORY (CS*)**
AboutMe.php	2.6%	Core	Stale	Stale	**Stale**		
BrowseCategories.php	8.7%	Edgeable					**Stale**
BrowseRegions.php	2.3%	Edgeable				**Stale**	
BuyNow.php	2.7%	Edgeable		Fresh			
PutBid.php	6.3%	Edgeable		Fresh			
PutComment.php	0.5%	Edgeable	Stale	Stale			
RegisterItem.php	0.6%	Edgeable		Fresh			
RegisterUser.php	1.2%	Core	Fresh				
SearchItemByCategory.php	18.2%	Core		**Stale**			
SearchItemByRegion.php	6.6%	Core		**Stale**			
StoreBid.php	4.7%	Edgeable		Fresh			
StoreBuyNow.php	1.4%	Core			**Fresh**		
StoreComment.php	0.5%	Core	Fresh				
UpdateItemQuantity.php	1.4%	Edgeable		Fresh			
ViewBidHistory.php	2.1%	Edgeable		Stale			
ViewItem.php	14.5%	Edgeable		Stale			
ViewUserInfo.php	3.4%	Edgeable	Stale				
html pages	16.3%	Edge caching					

Fresh cells indicate that the operations require fresh data from the cluster, and *Stale* indicates that stale state is fine too. Blue cells are the ones where operation requires single partition access, for shaded green cells operation requires multi partition access. (*) shows the output of the static analysis, i.e., policy for operations and clusters.

Table 5.1: RUBiS application static analysis.

Chapter 6

Evaluation

This chapter provides a detailed analysis of latency benefits achieved because of EdgeX's approach towards executing applications on a two-tier infrastructure. EdgeX's main task is to execute web operations at the edge sites whenever possible. To maximize this it needs to optimally place the replicas across all the sites. Therefore the focus of this analysis is on three main ideas, a) what are the latency overheads and benefits that can be achieved by executing a web operation at the edge site as compared to simply executing the operation at the core site, b) how effective is EdgeX at enabling edge execution of web operations, c) how does EdgeX's effectiveness change with geo-locality of the database.

6.1 Experimental setup

This section will provide details of the infrastructure used for evaluation of EdgeX and the benchmark application.

6.1.1 Infrastructure

The evaluation of EdgeX is performed on the SAVI Testbed [9], which consists of one core site and two edge sites. The core site is located in the Toronto region of Canada, and the two edge sites are located in the Vancouver (VC) region and the Ottawa (CT) region. Each site runs a virtual machine (VM) with Ubuntu64.2, 2 CPUs, 4GB RAM, and 40GB disk space. Each VM runs an Apache 2.2.22 web server and a MySQL 5.5.38 database service, therefore each site is capable of executing Apache web requests and database queries. The average latency between edge site VC and the core is 62 ms and between edge site CT and the core it is 7.5 ms. To simulate the workload the clients exist in two geographical regions, Vancouver and Ottawa. Clients in each region issue requests to their respective edge servers, and the average latency between them is less than a millisecond.

6.1.2 Benchmark

RUBiS application [21] is used as a benchmark to evaluate the performance of EdgeX. RUBiS is an eBay-like auction web application, with a relational database at the backend. The application has 17 web interactions which support web operations such as browsing items, viewing and putting bids on items, viewing and putting comments on items, buying and selling items, and registering users. It has 7 database tables: `USERS`, `ITEMS`, `CATEGORIES`, `REGIONS`, `BIDS`, `BUY_NOW`, `COMMENTS`.

RUBiS application supports three type of user sessions: visitor, buyer, and seller. Visitor session does not require any registration of the users, but it is limited to browsing. Buyer and seller sessions require user registration. During a buyer session, the user can put bids on items and view current bids, rating and comments. In the seller session, the user registers items and puts them up for sale. The benchmark provides a client-browser emulator, which can start multiple user sessions. Each session is a sequence of interactions separated by a certain think time. There are two kind of workloads, the browsing workload that consists of read-only interactions (only visitor sessions) and the bidding workload consists of 15% read-write interactions (mix of visitor, buyer, and seller sessions).

6.2 Benefit Vs. overhead

This section evaluates the latency benefits and overheads observed due to EdgeX's approach to operation execution. As each operation is received at the nearest edge site, metadata is consulted to decide if it should be executed locally or forwarded to the core site. The local execution and core site execution are compared to the base scenario where the operation issued by the client is sent directly to the core site for execution. The main idea is to evaluate the overhead caused due to checking the metadata and the benefit if the operation gets executed locally at the edge site.

Each edge location has a client that issues *ViewItem.php* web operations to view 100 distinct items and then calculates the average latency observed from that edge location.

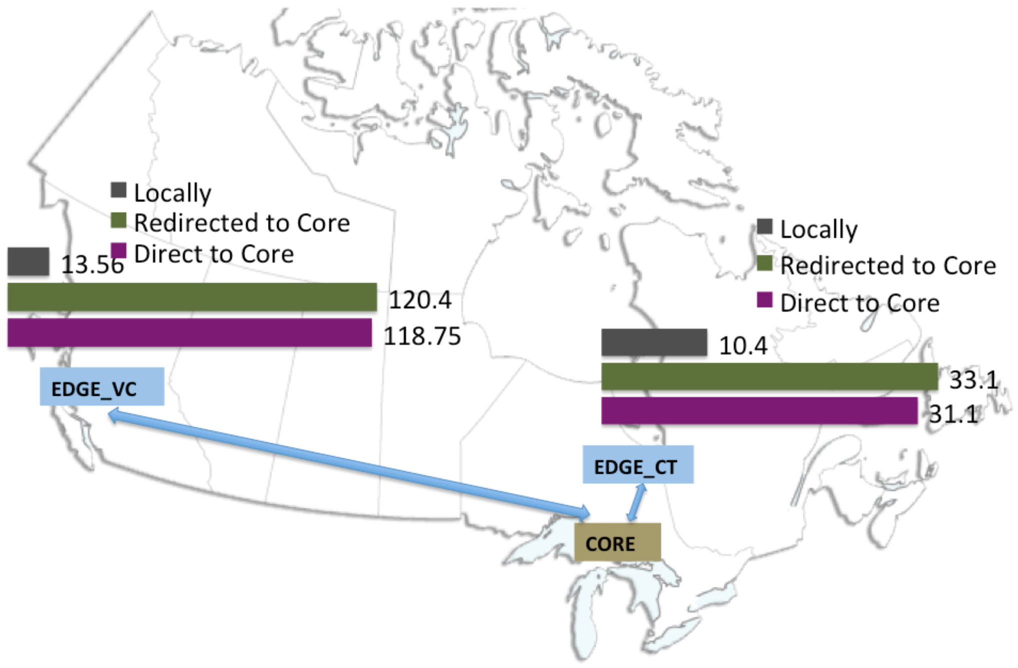


Figure 6.1: Shows the latency (in ms) for web operations from the edge sites when they are executed either directly at core, re-directed to core or executed locally at the edge site.

This experiment is repeated three times. On each run the ViewItem operations are executed in a different ways:

1. CORE, NO EdgeX: Here the client sends each operation directly to the core for execution, without using EdgeX. This is the base case and latency for this is around $118.75ms$ for edge VC and $31.1ms$ for edge CT.
2. CORE, WITH EdgeX: It assumes that EdgeX implementation is in place but the operation needs to be executed at the core site. The operation is received at an edge site and after the metadata check it is forwarded to the core site. The latency for this is on an average 4% higher than the base case due to the metadata check.
3. EDGE, WITH EdgeX: This assumes that EdgeX implementation is in place and the operation needs to executed locally at that edge site. The metadata check is performed and the operations fetches the data from the database partition located at that edge site. The latency for this case is much lower than the base case latency, 97% less for edge VC and 64% less for edge CT, because the *http* request does not have to go to the distant core site.

Figure 6.1 summarizes the results of this experiment with latencies (in ms) for the web operations issued from the two edge sites.

6.3 Best case benefits

The purpose of this experiment is to demonstrate how much benefit does EdgeX achieve in the ideal case for real applications. The ideal case is where all operations are issued from one edge site to maximize the geo-locality, and all the required partitions get migrated to the single best location.. It shows the difference in behavior of *edgeable* and *non-edgeable* operations at application runtime, and how much benefit is achieved in the average latency of the overall benchmark. Edge site VC is used because it is the farthest edge site, therefore the difference between the base case and the test scenario is clearly evident.

After the static analysis, tables `ITEMS` and `BIDS` have been grouped into `items` cluster, and tables `USERS` and `COMMENTS` have been grouped into `users` cluster, and all remaining tables are considered as individual clusters. Therefore, there are total of 5 clusters in the benchmark.

For this experiment, edge site VC executes the bidding mix workload. The workload has 500 clients each with multiple sessions of browsing, buying, bidding and selling items.

The experiment is performed in two steps. In the first step the complete workload is executed and EdgeX observes which partitions are required at the edge site and migrates them there. This prepares the system for the next step with all the partitions placed at the best location (either edge VC or core). In the next step the exact same workload from the first step is executed again. This time all the edgeable operations get executed at the edge site because their required partitions are already placed at the edge site. It is in the second step that the latency of each operation of the workload is noted, edgeable operations have very low latency as they get executed locally and the non-edgeable operations have higher latency as they get executed at the core.

Before evaluating the best case latency, it is important to have an idea about the distribution of the web operations in the benchmark, and their individual latencies. Table 6.1 gives an overview of the operations present in the benchmark and their percentage in the overall benchmark. It also shows which operations are edgeable and which execute at the core always, and how does the average latency for the operations varies from the base case. For this table the average latency of the first step is noted and reported in the column 'Latency with EdgeX'. This is not the best case latency because operations *eventually* start executing at the edge site as more and more partitions are replicated at the edge site. This gives an idea of the latency with EdgeX as compared to the base case when the replica placement at runtime starts from scratch. (Note, the latency of edgeable operations also depends on how long the workload was executed, it will reduce further with longer runs and more clients.) The average latency for most of the edgeable operations is lower than the base case, while it is slightly higher for the core web operations. For the overall benchmark, the average latency in the base case is $130.5ms$ and with EdgeX this latency drops to $77.6ms$

The edgeable operations are free to execute either at an edge site or the core site depending on the placement of replicas at the runtime. All the clusters that are assigned policy $core/static$ are completely replicated at all the edge sites beforehand, and clusters with policy $edge/dynamic$ have their partitions replicated at edge sites as per the workload at runtime. Figure 6.2, 6.3, and 6.4 show the latency of the workload in an ideal case of only one edge site. Each graph shows the latency for the base case and with EdgeX. In the base case, EdgeX implementation does not exist therefore all the operations are executed at the core. Figure 6.2 shows the latency of the operations that get executed at the core site. For example, operations *AboutMe*, *RegisterUser*, *SearchItemByCategory*, *SearchItemByRegion*, *StoreBuyNow*, and *StoreComment* get executed at the core site, and they all have latency higher than the base case because of the Apache routing via edge site. Figure 6.3 shows the latency of the operations that get executed at the edge site. These operations were executed at the edge site because the runtime system placed the required replicas at the edge site in step 1. For example, operations *BuyNow.php*, *PutBid.php*, *PutComment.php*, *RegisterItem*, *StoreBid*, *UpdateItemQuantity.php*, *ViewBidHistory.php*, and *ViewItem.php* are edgeable, and because their required replicas are available at the edge VC they get executed at this edge. Therefore, the latency for these operations is much lower than the base case. Figure 6.4 shows the latency of those requests that get executed at the edge site since the beginning of the workload because their required data has policy *core/static*. For example operations *BrowseCategoris.php*, *BrowseRegions.php*, *ViewUserInfo.php*, and *html* requests.

Table 6.1: Overview of RUBiS operations

Operation	**Percentage**	**Execution**	**Latency (ms) in base case**	**Latency with EdgeX**
AboutMe.php	2.6%	Core	171.1	164.1
BrowseCategories.php	8.7%	Edgeable	126.0	18.2
BrowseRegions.php	2.3%	Edgeable	118.9	11.5
BuyNow.php	2.7%	Edgeable	153.6	107.2
PutBid.php	6.3%	Edgeable	129.9	89.1
PutComment.php	0.5%	Edgeable	126.5	76.0
RegisterItem.php	0.6%	Edgeable	181.4	68.7
RegisterUser.php	1.2%	Core	118.3	179.6
SearchItemByCategory.php	18.2%	Core	121.9	122.7
SearchItemByRegion.php	6.6%	Core	151.1	160.8
StoreBid.php	4.7%	Edgeable	185.7	70.5
StoreBuyNow.php	1.4%	Core	182.0	134.8
StoreComment.php	0.5%	Core	183.4	191.5
UpdateItemQuantity.php	1.4%	Edgeable	185.1	67.2
ViewBidHistory.php	2.1%	Edgeable	124.8	115.0
ViewItem.php	14.5%	Edgeable	121.5	81.8
ViewUserInfo.php	3.4%	Edgeable	119.3	17.9
html pages	16.3%	Edge caching	116.4	9.9

6.4 Multiple edge workload

The job of replica manager is to observe the workload across all the edge sites and optimally place the replicas such that maximum edgeable operations get executed at edge sites.

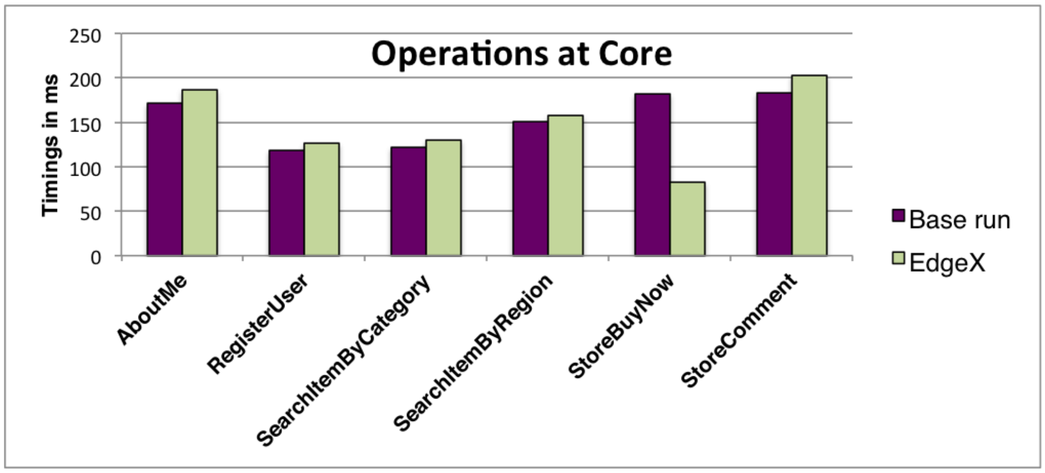


Figure 6.2: Comparision of latency for RUBiS workload operations executed at the Core site.

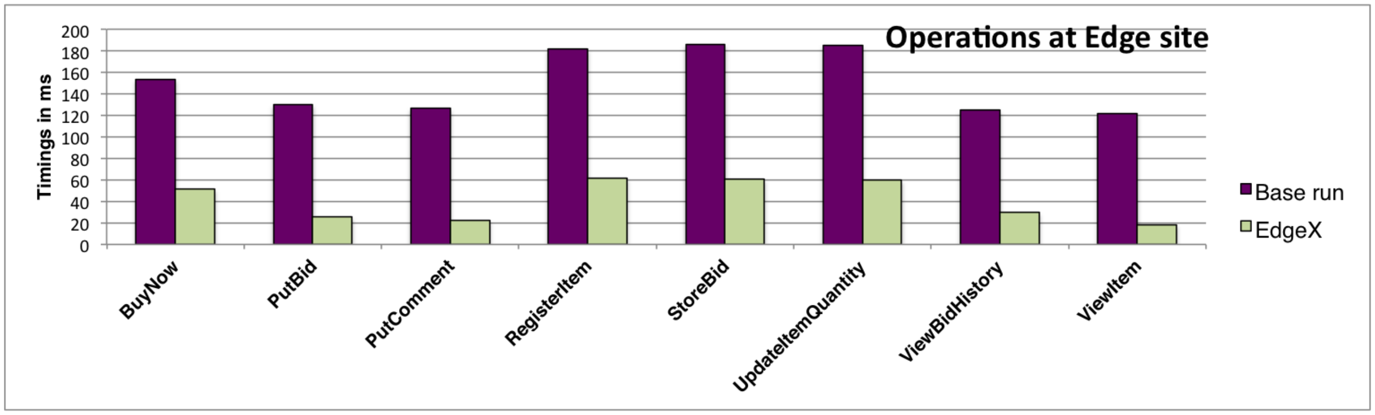


Figure 6.3: Comparison in latency for RUBiS workload operations executed at the Edge site.

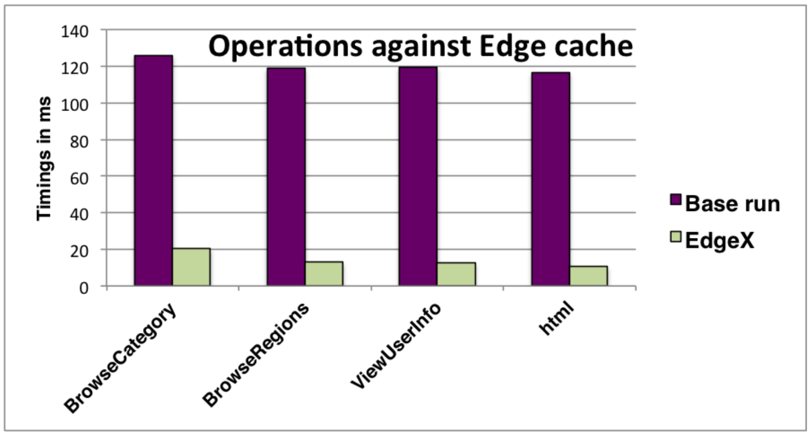


Figure 6.4: Comparison in latency for RUBiS workload operations executed at the Edge site.

Previous section demonstrated that when partitions have affinity to a single geographical site (one edge site) the latency for certain requests can reduce by 4x to 5x. However, it is possible that certain partitions are in demand from multiple edge sites, then the replica's location depends on the placement proposed by the cost model shown in section 4.2.1. This placement changes with changing workload, and when the placement changes there will be some web operations that will not be able to execute at the nearest edge sites. This section demonstrates how the latency varies when there is a contention for certain partitions from multiple edge sites.

The experiment uses two edge sites, VC and CT. Items in RUBiS belongs to its seller, and seller belong to an edge site, therefore each item belongs to a certain edge site. Therefore, both the edges have a certain set of associated items. Each edge consists of bidding workload with 500 clients, each client performs multiple sessions of browsing, bidding, buying and selling. In a single run of this experiment, clients at VC accesses CT items $x\%$ of times and its own items for the remaining times. Similarly, CT edge accesses VC items $x\%$ of times, and its own items otherwise. x is the parameter for this experiment. Each run consists of two steps. In the first step, the replica manager observes the complete workload executed at both the sites and at the end of it performs the calculations to identify an optimal site for each replica and places it there. In the next step, the same workload from both the edge sites is re-run. This time the latency of each web operation initiated at the VC edge site is noted. This experiment performs four runs, in each run the value of x is varied and variation in the latency is analyzed. Figure 6.5 shows this variation in latency for four runs and the base line for operations *BuyNow.php*, *PutBid.php*, and *ViewItem.php*,

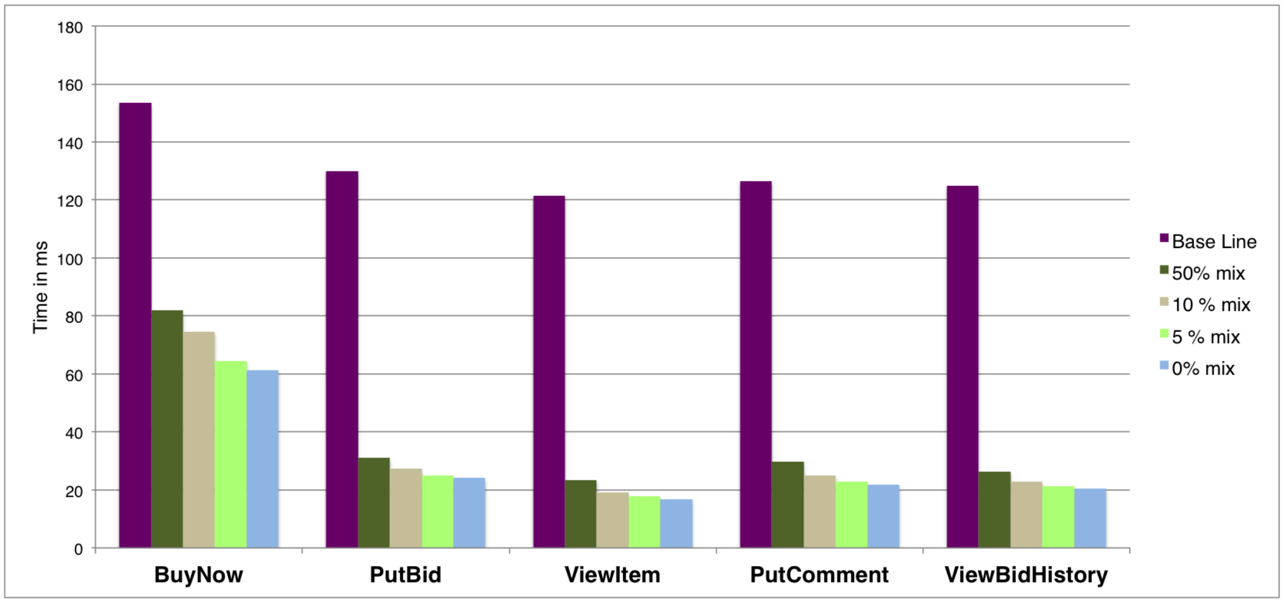


Figure 6.5: Impact on latency of VC initiated web operations when certain percentage of items are equally accessed from edge CT too.

PutComment.php, and *ViewBidHistory.php*. In the plot 0% mix bar shows the latency when VC edge accesses no items from CT edge site, 5% mix bar shows the latency when 5% of CT items are accessed at VC edge, similarly 10%, and 50% bars show the latency in their respective runs. The latency for the 50% mix is highest because maximum number of remote items are accessed, and it decreases with the decrease in percentage mix. It is important to note that even with 50% mix the performance is only on an average 35% worse compared to the ideal case of 0% mix and still around 3.5x better than the base line. It did not get 50% worse for 50% mix because out of the 50% some items would have been replicated at edge VC. Replica manager will do so because the benefit of placing them at edge VC will be greater than placing them at edge CT.

Chapter 7

Related Work

Executing web applications on a two-tier infrastructure has been widely studied. This section will discuss its related work. The discussion has been subdivided on three topics: *architecture* (section 7.1), *data partitioning* (section 7.2) and *replica management* (section 7.3).

7.1 Caching for web applications

Architectures for remotely caching database content first received significant attention in the 1980's amidst the research performed on client-server object-oriented databases. More recently, in past decade, this idea has been investigated for improving the scalability of web applications by delivering dynamic content from the mid-tier cache. Two accepted approaches for mid-tier database caching are *DBCache* [13] and *MTCache* [36]. Both these approaches assume an infrastructure similar to EdgeX where data is cached in a tier between the end user and the backend server and user always interacts with this middle tier. However the goal of these two systems differs from EdgeX. MTCache and DBCache's goal is to perform load balancing by offloading query execution to the middle tier, whereas EdgeX tries to exploit the geo-locality of the workload and execute as many requests as possible at the edge sites. Another difference between EdgeX and these two works is that EdgeX allows updates, insertions and fresh reads at the edge sites, whereas the other two systems execute those operations at the backend server.

Maintaining cache consistency has always been a challenge in a distributed mid-tier infrastructure. A popular approach to consistency maintenance has been to propagate updates from a central backend server [13, 14, 38]. Other sophisticated approaches such as exploiting query templates to maintain consistency has been proposed by Amiri et. al. [15], and distributed consistency management for edge caching has been proposed by Olston et. al. [40]. EdgeX relies on update propagation from a single primary site, but unlike previous approaches where the primary site is always a backend server, EdgeX picks a primary site at runtime depending upon the workload. Candan et. al. [20] have pursued the idea of invalidating cached materialized views to maintain consistency. They proposed algorithms to decide whether to invalidate cached views or not in response to update operations. Work by Yi Lin et. al [39] presents a scalable and transparent technique for maintaining the consistency of dynamic content in edge networks. Their approach is to update anywhere and execute all transactions at the local replica, however, their update transactions still require at least one wide area network message round, unlike EdgeX where most of the update transactions are executed locally at the nearest edge site.

Two popular industrial projects towards mid-tier caching are driven by IBM and Akamai Technologies. IBM's WebSphere [11] performs fragment caching [22, 26] to accelerate web applications which generate heterogeneous data. It also performs application offloading, similar to EdgeX, where application code can be executed at the caching servers. WebSphere suggests that some portions of the application, such as presentation tier and the business logic tier, should be offloaded to the edge servers and connect to the remaining application at the core server when necessary. Akamai Technologies [1] runs a leading content delivery network (CDN) with more than 175,000 servers in over 100 countries deployed in thousands of ISP networks. Akamai servers are responsible for caching customers' web contents, with configurable timeouts, at the edge of the network there by greatly reducing the latency of web operations. In present day web applications, where a typical HTML page consists of heavy static content such as images, video clips, and some dynamic (user generated) content, Akamai's CDN helps by offloading the sub-requests, fetching static content, to the edge servers. However, the dynamic content still needs to be retrieved from the database hosted at the central backend server. EdgeX's idea is to serve dynamic content, as well as the static content, from the edge of the network.

Routing the web requests depending upon the data location is an essential part of a distributed caching system. Different data can be present at different sites at different times. EdgeX maintains a mapping describing the location of replicas and consults it at runtime to route the requests. Certain works [47, 33] have advocated the use of hash functions to automatically assign data to servers. For EdgeX maintaining a mapping was a better choice because of changing workload patterns, assigning data according to a fixed hash function is not a feasible option.

7.2 Data partitioning

For EdgeX to execute an operation completely at a single site it is important that all of the required tuples be co-located at that site. To ensure this, EdgeX considers partitions as a collection of tuples from a group of tables which are co-accessed, and are related via foreign-key relationship. This idea of horizontally slicing a group of tables is borrowed from the data model of Google's Spanner project [24]. Similar to Spanner, the information about the group of co-accessed tables is explicitly provided by the application developer. This approach is also similar to caching materialized views defined by application code, which has been extensively used by most of the mid-tier caching projects [36, 13]. Recent work as JECB [45], by Khai Tran et. al. has shown that the task of deriving these partitions from the application code can be automated. Their partitioning goal aligns with EdgeX's - that is, partitioning to avoid multi-site transactions. They present a low-overhead data partitioning approach that analyzes the transaction source code of the given workload and the database schema to find a good partitioning scheme. JECB leverages partitioning by foreign-key relationships to automatically identify which tables should be grouped together; this makes JECB a suitable tool to automate the process of defining clusters for EdgeX. Oracle Database [8] is one of the commercial product that allows partitioning via foreign-key relationship. In the product it is featured as partitioning by reference or REF partitioning. The main focus of REF partitioning in Oracle database is to have efficient foreign-key joins between the tables.

Other work that focus on partitioning data to support OLTP workload includes Schism [25] and Horticulture [42]. Both of these works take a cost-based approach to minimize the number of distributed transactions in the system. Schism uses an initial training workload to form a *co-accessed* relationship graph between all the tuples, and then finds a *k-way* partitioning while minimizing the number of distributed transactions. Contrary to Schism's learning approach, Horticulture analyzes the database schema, the structure of the application's stored procedures, and a sample transaction workload, and then automatically generates partitioning strategies that minimizes the number of distributed transactions. As EdgeX does not allow distributed transactions, minimizing the number of distributed transactions is not enough for EdgeX's implementation. It needs a reliable partitioning scheme where each partition consists of all the tuples that are co-accessed. Therefore, an approach like JECB is closer to the automated partitioning tool required for EdgeX.

7.3 Replica management

One of the major tasks of EdgeX is to automatically move the replicas with the changing workload at application runtime. The problem of automatic replica placement depending upon the workload is well studied in the distributed systems community [17, 23, 16, 31, 35, 43, 48, 32]. Different works have studied different flavors of this problem. Some solved the problem of load balancing within a data centre [16, 31, 35, 43, 48], some solved the problem of meeting SLAs in a geo-distributed network [17] or the problem of minimizing the bandwidth for synchronization [32]. Some have discussed the granularity of replication [23]. The goal of EdgeX's replica manager is similar to managing replicas in a geo-distributed system where latency between sites is significant. Like Ardekani et. al. [17], EdgeX takes a cost-based approach to decide a location for each primary and secondary replica. This cost model takes latency between sites, demand from each site, and admin-defined constraints, such as the maximum number of allowed replicas, into consideration. Similar to PNUTS's records [23], EdgeX has partitions, and in both these works the replication is done at partition level. Partitions are independent of one another, i.e., the location of one partition does not impact the location of other partitions. This allows the replica manager to move replicas as per the workload requirements.

Kadambi et al. [32] solve a variant of replica management problem with the goal of minimizing the bandwidth consumed in keeping replicas in sync and forwarding reads to remote sites. EdgeX's goal partially aligns with their's, that is to send minimum read operations to remote sites. They extend the per-record selective replication from PNUTS [23] and their placement algorithm is derived from Wolfson et al.'s work [48]. Agarwal et al. propose a system called Volley [12] that performs automated replica placement with the aim of balancing datacenter load, reducing inter-datacenter traffic and client latency. Like EdgeX, Volley also computes the data placement utility at regular intervals and move replicas as required, however the granularity of replication is a tablet, unlike record or a partition. Tran et al. propose a system called Nomad [46] that allows data migration to happen at the granularity of *overlays*, which are abstractions holding object containers across datacentres. Nomad tries to predict the user movement from the trace of access logs and places the overlays at the appropriate sites, whereas EdgeX tries to follow the user in case the user changes its location.

In EdgeX updates are propagated lazily from single primary replica to multiple secondary replicas. However, there are other concurrency control methods for geo-distributed databases such as two-phase commit protocols [30] or optimistic concurrency control [18]. Asynchronous replication has been widely used in distributed database system to achieve scalability while relaxing strong consistency. Master-slave replication is a typical model for asynchronous replication [27, 41]. Both PNUTS and EdgeX have extended the master-slave replication model at the granularity of a record and partition, respectively. The mastership is dynamic and changes with workload. Other systems such as Dynamo [28] have used a gossip based protocol to eventually reconcile the state of data from multiple master sites.

Chapter 8

Conclusion and Future Work

EdgeX is a solution towards caching static as well as dynamic content close to the user for better web application performance, latency in particular. For OLTP style applications, such as *eBay*, deployed on two-tier infrastructure, EdgeX proposes a data partitioning, data replication, and application execution scheme such that most of the user interactions can be handled at the edge of the network. For a given web application, EdgeX performs two tasks: it first identifies which database tables are safe to be partitioned and place at the edge sites at runtime, and which web operations can execute completely at an edge site. Second, at application runtime it monitors the workload pattern at regular intervals and places the replicas such that most of the edgeable web operations can find the required data at the nearest edge site. While doing these tasks, EdgeX's aim is to not just serve read-only data at the edge, whereas handle write and update operations as well at the edge sites. This is one of the unique features of EdgeX when compared to other mid-tier caching systems. EdgeX's implementation can only guarantee eventual consistency but the consistency model is flexible enough to guarantee fresh data for certain operations as demanded by the application developer. It provides application level consistency, where each web operation has different consistency requirements and guarantees.

The evaluation of EdgeX with RUBiS benchmark has shown that on an average for a read-write workload, upto 70% of the web operations can be executed at the edge site. The actual percentage of operations that get executed at edge sites depends on the geo-locality of the workload. By moving the replicas at an edge site, EdgeX can achieve on an average 4x latency benefit for edgeable operations, this number was obtained when RUBiS benchmark was executed on SAVI Testbed with EdgeX's implementation. EdgeX is efficient in placing the replicas at right locations even when there is a competition on certain replicas from multiple locations. In the worst case where almost 50% of the replicas get equally accessed from multiple edge sites, the performance suffered only 34% when compared to the ideal case where no replicas are shared between sites. This still was 3.5x better than the base line where everything gets executed at the core site.

Taking this work forward, there exists some new challenges that can add to the EdgeX's capabilities. First one is to automate the process of partitioning. Right now a partition is defined by the application developer, to reduce this burden, the task of detecting partitions can be automated. For example, by integrating a tailored version of JECB [45]. With automated partitioning, EdgeX can just look at the database schema and the application code to detect the right partitioning for the application. Second extension can be to constraint the runtime replica placement on size. In the current implementation, it is the application developer who declares the limit on number of replicas allowed per partition. EdgeX should self-detect the least number of replicas required for each partition to achieve maximum latency benefit under a given space limit. Limiting the number of replicas will be useful because edge sites have lower storage capacity compared to the central core site, and EdgeX should automatically decide the top most relevant partitions for a given site, and for a given size constraint, based on the workload.

References

[1] Akamai technologies.
https://www.akamai.com/.

[2] Amazon.
http://www.amazon.com/.

[3] Amazon web services global infrastructure.
http://aws.amazon.com/about-aws/global-infrastructure/.

[4] ebay.
http://www.ebay.com/.

[5] Google data center locations.
http://www.google.ca/about/datacenters/inside/locations/.

[6] Groupon.
http://www.groupon.com/.

[7] *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.

[8] Partitioning with oracle database 11g release 2.
http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-partitioning-11gr2-2011-12-1392415.pdf/.

[9] Savi testbed.
http://www.savinetwork.ca/.

[10] SymmetricDS.
http://www.symmetricds.org/.

[11] WebSphere.
http://www-01.ibm.com/software/ca/en/websphere/.

[12] Sharad Agarwal, John Dunagan, Navendu Jain, Stefan Saroiu, Alec Wolman, and Harbinder Bhogan. Volley: Automated data placement for geo-distributed cloud services. In *Proc. of the 7th USENIX Conference on Networked Systems Design and Implementation*, pages 2–2, Berkeley, CA, USA, 2010. USENIX Association.

[13] Mehmet Altinel, Qiong Luo, Sailesh Krishnamurthy, C. Mohan, Hamid Pirahesh, Bruce G. Lindsay 0001, Honguk Woo, and Larry Brown. Dbcache: database caching for web application servers. In *Proc. ACM SIGMOD*, page 612. ACM, 2002.

[14] Khalil Amiri, Sanghyun Park, and Renu Tewari. Dbproxy: A dynamic data cache for web applications. In *Proc. ICDE*, pages 821–831, 2003.

[15] Khalil Amiri, Sanghyun Park, Renu Tewari, and Sriram Padmanabhan. Scalable template-based query containment checking for web semantic caches. In *Proc. ICDE*, pages 493–504, 2003.

[16] Khalil Amiri, David Petrou, Gregory R. Ganger, and Garth A. Gibson. Dynamic function placement for data-intensive cluster computing. In *Proc. of the Annual Conference on USENIX Annual Technical Conference*, ATEC '00, pages 25–25, Berkeley, CA, USA, 2000. USENIX Association.

[17] Masoud Saeida Ardekani and Douglas B. Terry. A self-configurable geo-replicated cloud storage system. In *Proc. 11th USENIX Symposium on Operating Systems Design and Implementation*, pages 367–381, 2014.

[18] Philip A. Bernstein and Nathan Goodman. Timestamp-based algorithms for concurrency control in distributed database systems. In *Proc. of VLDB*, pages 285–300. VLDB Endowment, 1980.

[19] Carsten Binnig, Abdallah Salama, Alexander C. Müller, Erfan Zamanian, Harald Kornmayer, and Sven Lising. Xdb: A novel database architecture for data analytics as a service. In *Proc. of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 39:1–39:2, New York, NY, USA, 2013. ACM.

[20] K. Selçuk Candan, Divyakant Agrawal, Wen-Syan Li, Oliver Po, and Wang-Pin Hsiung. View invalidation for dynamic content caching in multitiered architectures. In *Proc. VLDB*, pages 562–573, 2002.

[21] E. Cecchet et al. Performance and scalability of EJB applications. In *Proc. ACM OOPSLA*, pages 246–261, 2002.

[22] Jim Challenger, Paul Dantzig, and Arun Iyengar. A scalable and highly available system for serving dynamic data at frequently accessed web sites. In *Proc. of the ACM/IEEE Conference on Supercomputing*, page 47, 1998.

[23] Brian F. Cooper et al. PNUTs: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1(2):1277–1288, August 2008.

[24] James C. Corbett et al. Spanner: Google's globally-distributed database. In *Proc. of OSDI*, pages 261–264, October 2012.

[25] Carlo Curino, Evan Jones, Yang Zhang, and Sam Madden. Schism: A workload-driven approach to database replication and partitioning. *Proc. VLDB Endow.*, 3(1-2):48–57, September 2010.

[26] Anindya Datta, Kaushik Dutta, Helen M. Thomas, Debra E. VanderMeer, Suresha, and Krithi Ramamritham. Proxy-based acceleration of dynamically generated content on the world wide web: an approach and implementation. In *Proc. ACM SIGMOD*, pages 97–108, 2002.

[27] Khuzaima Daudjee and Kenneth Salem. Lazy database replication with snapshot isolation. In *Proc. of VLDB*, pages 715–726. VLDB Endowment, 2006.

[28] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proc. of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, pages 205–220, New York, NY, USA, 2007. ACM.

[29] John Dilley et al. Globally distributed content delivery. *IEEE Internet Computing*, 6(5):50–58, September 2002.

[30] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 1992.

[31] Galen C. Hunt and Michael L. Scott. The coign automatic distributed partitioning system. In *Proc. of OSDI*, OSDI '99, pages 187–200, Berkeley, CA, USA, 1999. USENIX Association.

[32] Sudarshan Kadambi, Jianjun Chen, Brian F. Cooper, David Lomax, Raghu Ramakrishnan, Adam Silberstein, Erwin Tam, and Hector Garcia-Molina. Where in the world is my data? *PVLDB*, 4(11):1040–1050, 2011.

[33] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proc. of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, pages 654–663, New York, NY, USA, 1997. ACM.

[34] Ron Kohavi and Roger Longbotham. Online experiments: Lessons learned. *Computer*, 40(9):103–105, 2007.

[35] Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Placement algorithms for hierarchical cooperative caching. In *Proc. of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, pages 586–595, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[36] Per-Åke Larson, Jonathan Goldstein, and Jingren Zhou. MTCache: Transparent mid-tier database caching in SQL Server. In *Proc. ICDE*, pages 177–189, 2004.

[37] Tom Leighton. Improving performance on the internet. *Commun. ACM*, 52(2):44–51, February 2009.

[38] Wen-Syan Li, Oliver Po, Wang-Pin Hsiung, K. Selçuk Candan, Divyakant Agrawal, Yusuf Akca, and Kunihiro Taniguchi. Cacheportal II: acceleration of very large scale data center-hosted database-driven web applications. In *Proc. of VLDB*, pages 1109–1112, 2003.

[39] Yi Lin, Bettina Kemme, Marta Patiño-Martínez, and Ricardo Jiménez-Peris. Enhancing edge computing with database replication. In *26th IEEE Symposium on Reliable Distributed Systems*, pages 45–54, 2007.

[40] Christopher Olston, Amit Manjhi, Charles Garrod, Anastassia Ailamaki, Bruce M. Maggs, and Todd C. Mowry. A scalability service for dynamic web applications. In *CIDR*, pages 56–69, 2005.

[41] Esther Pacitti, Pascale Minet, and Eric Simon. Fast algorithms for maintaining replica consistency in lazy master replicated databases. In *Proc. of VLDB*, pages 126–137, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[42] Andrew Pavlo, Carlo Curino, and Stanley B. Zdonik. Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems. In *Proc. ACM SIGMOD*, pages 61–72, 2012.

[43] Gokul Soundararajan, Cristiana Amza, and Ashvin Goel. Database replication policies for dynamic content applications. In *Proc. of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems*, EuroSys '06, pages 89–102, New York, NY, USA, 2006. ACM.

[44] Douglas Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos Aguilera, and Hussam Abu-Libdeh. Consistency-based service level agreements for cloud storage. In *Proc. ACM Symposium on Operating Systems Principles.* ACM, November 2013.

[45] Khai Q. Tran et al. JECB: A join-extension, code-based approach to OLTP data partitioning. In *Proc. ACM SIGMOD*, pages 39–50, 2014.

[46] Nguyen Tran, Marcos K. Aguilera, and Mahesh Balakrishnan. Online migration for geo-distributed storage systems. In *Proc. of the USENIX Conference on USENIX Annual Technical Conference*, pages 15–15, Berkeley, CA, USA, 2011. USENIX Association.

[47] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, and Carlos Maltzahn. Crush: Controlled, scalable, decentralized placement of replicated data. In *Proc. of the 2006 ACM/IEEE Conference on Supercomputing*, New York, NY, USA, 2006. ACM.

[48] Ouri Wolfson, Sushil Jajodia, and Yixiu Huang. An adaptive data replication algorithm. *ACM Trans. Database Syst.*, 22(2):255–314, June 1997.