Tiny URL Pilot Project

**System Design Document**

Version 0.2

Date Baselined:

Last Revised: 03/06/2018

| Prepared by: | For further information, please contact: |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
| **Low Level Design Specification Document** | |
| **Project Name:** |  |
| **Project Identifier:** |  |

|  |  |
| --- | --- |
| **Project Manager (Business):** |  |
| **Project Manager (IT):** |  |
| **Solutions Delivery Manager:** |  |

Revision History

| Version No. | Date | Author | Revision Description |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Low Level Design Approvals

| Approvals | Name/Signature | Date Approval Received |
| --- | --- | --- |
| Solutions Architect (HLD Author) |  |  |
| Systems Analysis Center of Excellence  **(required only if there was no HLD accompanying this design)** |  |  |
| Data Analysis Center of Excellence |  |  |
| Solution Delivery Manager |  |  |
| Impacted System &  System Owner | NA | NA |

Document Change Management and Distribution Procedure

Copies of this document will be considered as drafts, until this document has been reviewed and baselined in accordance with the defined process.

Requests to change a baselined version of this document must be submitted and handled in accordance with the defined process.

New versions of this document are to be distributed to appropriate staff or made accessible via the project repository for reference. Notification of a new version must be communicated to appropriate individuals.

Document Instructions

.

Tailoring Guidelines

Contents

[Introduction 4](#_Toc516041442)

[What’s Tiny URL 5](#_Toc516041443)

[Tiny URL- Design Approach 5](#_Toc516041444)

[Traffic: 5](#_Toc516041445)

[URL length: 5](#_Toc516041446)

[Caching: 6](#_Toc516041447)

[Vertical Scalability 6](#_Toc516041448)

[Horizontal Scalability 6](#_Toc516041449)

[In-Process Caching 6](#_Toc516041450)

[In-Memory Database: 7](#_Toc516041451)

[In-Memory Distributed Caching 7](#_Toc516041452)

[Major Technologies Used 7](#_Toc516041453)

# 

# Introduction

Scope of the document is to detail the adopted approach to solve the Tiny URL assignment and also to illustrate various techniques to revamp the scalability and performance portability.

# What’s Tiny URL

Basically, TinyURL is a URL shortening service, a web service that provides short aliases for redirection of long URLs. There are many other similar services like Google URL Shortener, Bitly etc..

For example, URL http://blog.gainlo.co/index.php/2015/10/22/8-things-you-need-to-know-before-system-design/ is long and hard to remember, TinyURL can create a alias for it – http://shorturl.com/j7ve58y. If you click the alias, it’ll redirect you to the original URL.

# Tiny URL- Design Approach

As a part of design approach, we need to consider the following points

## **Traffic**:

Twitter has about 300M active users per month. Let's assume we are 10% popular as Twitter and each user generates 1 shortened URL per day. This leads to 30M service calls per month (1M calls/day). If we are going to keep our service for 5 years, our service will generate about 1.8B records. In order to retain that much massive data for our Tiny URL System, NoSQL databases are well suited with more scalable and provide superior performance.

For Tiny URL assignment purpose, I used Mongo DB. Mongo DB is a NoSQL document database that has a master-slave architecture. When creating a cluster and scaling it, we create and manage a [sharded cluster](https://docs.mongodb.com/manual/tutorial/deploy-shard-cluster/) and a [replication set](https://docs.mongodb.com/manual/tutorial/deploy-replica-set/), which is another name for a replication cluster. Mongo DB uses sharding to support deployments with very large data sets and high throughput operations.

**URL length:**

Shortened URL can be combinations of numbers (0-9) and characters (a-Z). If we have shortened URL length as k, it gives us 62^k unique combinations. Let's set k = 6, this will render 56.8B combinations which is far more than what we need. Following snippet of code was used to create Shortened URL

**public** **static** **final** String ***ALPHABET*** = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

**public** **static** **final** **int** ***BASE62*** = ***ALPHABET***.length();

/\*\*

\* Returns the base 62 string of a long.

\* **@param** a long.

\* **@return** the base 62 string of a long

\*/

**public** **static** String toBase62(**long** value) {

**final** StringBuilder sb = **new** StringBuilder(1);

**do** {

sb.insert(0, ***ALPHABET***.charAt((**int**) (value % ***BASE62***)));

value /= ***BASE62***;

} **while** (value > 0)

**return** sb.toString();

}

And Folllowing piece of code is to reverse to original url

/\*\*

\* Returns the base 62 value of a string.

\* **@param** a string.

\* **@return** the base 62 value of a string.

\*/

**public** **static** **long** toBase10(String str) {

**return** *toBase10*(**new** StringBuilder(str).reverse().toString().toCharArray());

}

**private** **static** **long** toBase10(**char**[] chars) {

**long** result = 0;

**for** (**int** i = chars.length - 1; i >= 0; i--) {

result += *toBase10*(***ALPHABET***.indexOf(chars[i]), i);

}

**return** result;

}

**Caching:**

Improves performance since data does not have to be retrieved again from the original source. Cache can be designed on single/multiple JVM or clustered environment. Different scalability scenarios where caching can be used to meet the nonfunctional requirement are as follows.

## Vertical Scalability

This can be achieved by upgrading a single machine with more efficient resources (CPU, RAM, HDD, and SSD) and implementing caching. But it has limitations in regards to upgrading a cache up to a certain limit. In the below use case, the application’s performance can be enhanced by adding more memory and implementing caching at the application level.

## Horizontal Scalability

This can be achieved by adding more machines and implementing caching at the application level on each machine.

## In-Process Caching

In-process caching enables objects to be stored in the same instance as the application, i.e. the cache is locally available to the application and shares the same memory space.

Here are some important points for considering in-process caching:

* If the application is deployed only in one node, i.e. has a single instance, then in-process caching is the right candidate to store frequently accessed data with fast data access.
* If the in-process cache will be deployed in multiple instances of the application, then keeping data in sync across all instances could be a challenge and cause data inconsistency.
* If server configurations are limited, then this type of cache can degrade the performance of any application since it shares the same memory and CPU. A garbage collector will be invoked often to clean up objects that may lead to performance overhead. If data eviction isn't managed effectively, then out-of-memory errors can occur.

## In-Memory Database:

This type of database is also called a main memory database. It's where data is stored in RAM instead of a hard disk to enable faster responses. Data is stored in compressed format with good SQL support.

For this assignment purpose, Mongo DB modelling is as below

Id

Original URL

Sequence

Created On

## In-Memory Distributed Caching

Distributed caches (key/value objects) can be built externally to an application that supports read/write to/from data repositories, keeps frequently accessed data in RAM, and avoid continuous fetching data from the data source. Such caches can be deployed on a cluster of multiple nodes, forming a single logical view. Caching clients use hashing algorithms to determine the location of an object in a cluster node.

Here are some important points for considering distributed caching:

* An in-memory distributed cache is the best approach for mid- to large-sized applications with multiple instances on a cluster where performance is key. Data inconsistency and shared memory aren't matters of concern, as a distributed cache is deployed in the cluster as a single logical state.
* As inter-process is required to access caches over a network, latency, failure, and object serialization are some overheads that could degrade performance.

# Major Technologies Used

* JDK 1.8
* Spring Boot 1.5.9
* Mongo DB 3.2
* Junit