Contents

Page.

Introduction

Nowadays innovations are growing fast and existing technologies are evolving. Hyperloop, exploring the universe, science research work, green systems and more daily, but so needed things, such as transport, smart homes, communication technologies... The fact, that they require more power, space, flexibility, reliability and speed, should not be overlooked. One of important component for lots of kinds of such innovations is fast and reliable storage. In 21th century the term ”distributed datastore” became habitual for usage. And some distributed datastores are growing, some do not. Why? There is a reason that have a great impact on fast growth of such a datastore, when they lose consistency value as soon as they become larger. For some of systems it is very important part of their stable work.

There is a well-know theorem, called CAP, that claims that it is impossible to satisfy fully consistency, avaliability and partition tolerance value simultaneously.

We do not argue this theorem, but what we do - try to get round this problem. We explain the idea in detail in the first chapter.

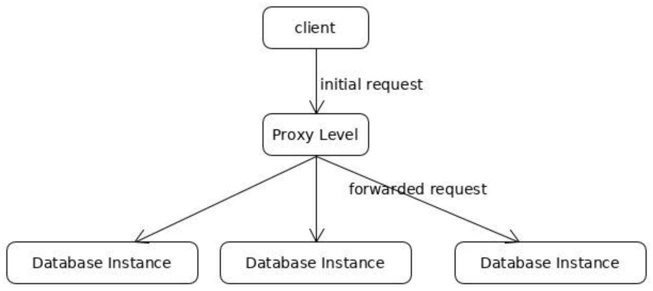
Chapter 1

Hypothesis: Distributed Datastore: Handle request consistent

, load balancer, distributed hash table It is well-known fact that according CAP-theorem it is impossible to support strong consistency and not to lose availability.

What if to try to get round this problem by handling request to database by the list of consistent nodes for. appropriate to the dataunit incoming in the request. We want to estimate how it may impact availability and partition tolerance if we want to get round consistency issue this way.

There are several solutions. But they all have general architecture: (Picture of architecture)



This architecture may have several implementations:

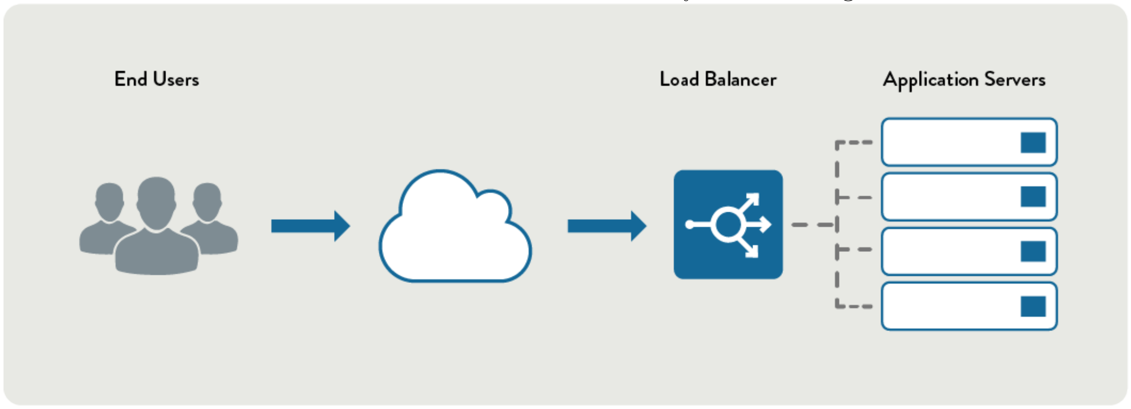
- load balancer level - load balancer may include or implement this solution
- hybrid: application level - it may be done as an own separate algorithm working on the top of request handling.
- manual redirection between nodes at the database level

We want to compare the architecture value of all of these solutions and propose the best one. In the next section we consider load balancer level solution, in the section 2 - hybrid and in the section 3 - manual redirection at database level.

Chapter 2

Load balancer handling solution and its estimates

Firstly let us remind what load balancer is. From [?] load balancing is the technique that distributes the workload across multiple devices. Generally it stands behind several web servers. It is most relevant to describe load balancer by the following scheme:



Load balancer distributes traffic by different mechanisms.
They are :

Round robin	default load balancing method. Round Robin mode passes each new connection request to the next server in line, eventually distributing connections evenly across the array of machines being load balanced.
Weighted least	
Ratio	The BIG-IP system distributes connections among pool members or nodes in a static rotation according to ratio weights that you define. In this case, the number of connections that each system receives over time is proportionate to the ratio weight you defined for each pool member or node. You set a ratio weight when you create each pool member or node.
Dynamic Ratio	The Dynamic Ratio methods select a server based on various aspects of real-time server performance analysis. These methods are similar to the Ratio methods, except that with Dynamic Ratio methods, the ratio weights are system-generated, and the values of the ratio

	weights are not static. These methods are based on continuous monitoring of the servers, and the ratio weights are therefore continually changing
ode) Fastest (application)	The Fastest methods select a server based on the least number of current sessions.
Least Connections	The Least Connections methods are relatively simple in that the BIG-IP system passes a new connection to the pool member or node that has the least number of active connections. .....

The way load balancer distributes traffic, lie on the hypothesis in the perfect way (see previous chapter).

Propose architecture (scheme).

Haproxy requires less additional feature to support our needs or even already implements them.

Consistent hashing algorithm or algorithm custom that takes list of available nodes from the request and load balance between them.

Focus on the state of the node.

The purpose of our load balancing technique is to redirect request to nodes that are already consistent making decision on the request.

Let's focus on dynamic ratio method.

Conclusion for this section: Nowadays there is lack of load balancers implementation that can support our need. But a lot of them are opensource solutions and may be extended soon.

How many nodes at maximum load balancer supports

Chapter 3

Hybrid algorithm

Other is described in next section (Algorithm)

Chapter 4

Own load balancing algorithm estimates

Possibility to add it to load balancer Where to get consistent nodes list from.
(Hash Table? masy be. another db? no. separate API? it anyway will take list from dynamic dict.)
so hash table to store nodes, its structure, evaluation of requests

Chapter 5

Distributed hash table

What is distributed hash table and its implementations
The maximum size of hash table
Performance of different storages (redis, ...).
Formulas to keep it stably fast. Formula for number of nodes for DDS and number of keys in hash table (numbers of dataunits).
Hash Table performance (read/write requests per second) - data from site, performance etc.

Chapter 6

Imitation model to communicate at load balancer level

Architecture, diagrams, or experiments with algorithm sequence , activity diagrams for all three solutions.
To redirect over consistent nodes on API layer is a solution that delays response a lot. Request come to not consistent node then redeirected to consistent node, but it is not alive, then eventually come to the proper node. This solution is best to implement on load balancer algorithm level: get the consistent nodes and load balance between them by any enabled default algorithm (round robin, least response time, ...)
But estimate both solutions.
solutions.

Chapter 7

Mathematical Model of ...

and model to increase consistency value.

Let we have distributed datastore with such characteristics:

N	is a finite set of nodes of a distributed datastore;
L	is a finite set of links of a distributed datastore;
$\partial : L \to 2^N$	is a mapping that associates each link with two nodes that it binds;
D	is a finite set of stored data units;
$r : D \to 2^N$	is a mapping that associates each data unit d with a subset of nodes that store its replica;
N_d	is a finite set of nodes that are having given dataunit d;
$l(N_d)$	is a number of nodes in datastore that are having given dataunit;
n_c	is a number of nodes in a subset of N_d where all nodes have the same replica.

Chapter 8

Estimates, time complexity of algorithm basing on lb balancer, distributed hash table

Chapter 9

Appendix: Load balancer, distributed hash table complexity estimates

BIBLOGRAPHY

1. Kyrylo Rukkas, Galyna Zholtkevych: Distributed Datastores: Towards Probabilistic Approach for Estimation of Dependability, ICTERI 2015: 523-534
2. P.P. Geethu Gopinatha, Shriram K.Vasudevan: An In-depth Analysis and Study of Load Balancing Techniques in the Cloud Computing Environment, Procedia Computer Science Volume 50, 2015, Pages 427-432 https://www.sciencedirect.com/science/article/pii/S1877050915005104 https://devcentral.f5.com/articles/what-is-load-balancing-24740
3.