OVERVIEW

From the description of the task is clear, that the system should be accessible from any point of the world, besides, desirable to receive a page with a button and a number during 300 ms or less (when the response comes during 300 milliseconds or less the user is not a feeling that he was waiting). And read requests expected to more than write requests.

Because of the geographic distribution of users is best to use a geographically distributed server cluster.

In the task nothing is said about the requirements of data accuracy. I assumed that each user does not need to obtain the most recent value of the counter, for example, will be enough to show him the value of the counter, which was actually 2 seconds ago. In this case, we can use a cluster with the master and its mirror in the main data center and 5th geographically distributed slave - one on each continent, except a continent with the master and its mirror and Antarctica. Master will process the write requests and replicate counter data to other servers in the cluster - this is the most productive node in the cluster. The mirror can be a slave or be a second master, as the host write requests. Thus, there are 2 ways to use it:

1. The master works on record, the mirror is only for reading; replication: master-slave.
   In this case, when master is falling the mirror becomes a new master. Data that did not have time to replicate in the mirror may be lost, but can be saved by using bidirectional replication (BDR).
   Fallen master does not accept requests while raising, all of its customers are beginning to turn to the mirror, user sessions are lost.
   A software in this case, without bidirectional replication:
   • for the master and to the left: Debian + nginx with redis extension + redis;

   with bidirectional replication:
   • for the master: Debian + nginx with postgresql and redis extensions + postgresql + redis;
   • for the slaves: Debian + nginx with postgresql extensions + postgresql.

   PostgreSQL is used to store a counter and is introduced due to the fact that the tool supports BDR while Redis is not support BDR.

2. Both nodes work on reading and writing; replication: master-master. Load balancing method between them: Sticky Sessions.
   Software (masters and slaves): Debian + nginx with redis extension + redis.
   Using bidirectional replication can improve performance, but then it may happen that none of the masters will never contain completely current data. Refuse this way.

The first method appears preferable, as it gives the increase of write performance, both approaches give equally good reliability (assuming the use of bi-directionally replicate data repository in the first method). In turn, using Redis as a primary storage in the first approach we will give more write and read performance, but, unfortunately, allow the probability of losing a small amount of transactions when the master is fall. Next we will consider both ways.

In the case of the fall of the master and mirror, one of the slaves must take a master's duties.

Slaves will be distributed geographically, and will handle read-only queries from its continents. Its data can be updated with a delay of replication, thus not always all users will see the latest value of the counter.

If a slave is falling, it temporarily disconnecting from the processing of requests.

When adding a new slave in cluster, this slave must get data from the master and starts to work (making at DNS server).

Requests for authorization, as well as requests to increment the counter are always processed only in master (masters), slaves will be stored a value of the counter only.

Requests routing should be made at the DNS level by using Anycast technology.

If the relevance is very important, we can use a distributed master-master or not to distribute the cluster nodes geographically - keep a couple of servers in one data center and hope to noncritical delays for users from Australia.

If the relevance of the data on each node is required, we will have to use a distributed master-master or not to distribute the cluster nodes geographically - keep two servers one data center and hope to noncritical delays for users in Australia.

It is worth noting that the problem of distribution and replication of data relate only to storage. Software and configuration files are duplicated on all machines.

With regards to hardware, the basic conditions:

1. Combine drives of each node in a hardware RAID-arrays with battery (BBU), for example, with level 10.

2. It is desirable to have an amount of RAM that can accommodate all of the necessary data for each node.

THE ARTCHITECTURE OF NODES

Reading of request comes at a node is processing using Nginx in 4 steps: page's layout is receiving from a file, receives the counter value from Redis or PostgreSQL, it brings them

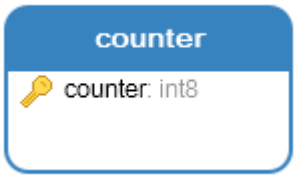
together and gives the client. Two initial steps can run in parallel, "mixing" of the counter and the page template can be made by SSI technology, for example.

When users is registering, the Redis storage writing a key-value record, key is a hash value of the concatenation of login, a user password and a salt that contains a any non-integer value. The hash of the login and password comes from client, and salt are added later - during the formation of the request to the Redis for adding a record about new user. In this record we set value to true and authorizing the user by cookie.

Further authentication takes place on the condition, that the authorization sequence was saved in Redis store.

There is also a safer option: server requests a username and password only when the user attempts to increase the value of the counter. In this case, a hash of the login and password come with a request for the increment, the server checks the credentials and in case of their legitimacy increments the counter, cookie is not set, and the next attempt to increment the counter, system again prompts the user to input a login and password. However, that this method is at a higher security it may be not convenient from the standpoint of user experience.

If for the counter used PostgreSQL database, we will need to create a table with one column and one line, resulting in a single cell and will store the counter value. It looks very strange, but it is a payment for the use of BDR.



If the counter is stored in the Redis, everything becomes easier - we simply write the value by key named, for example, “counter” in the storage.

The recording increment the counter in the storage all the time is accessible only to authorized users - those who sent legitimate hash (by cookie or not) and those who updated the counter for more than 24 hours ago, the time of the last increment by the user server can get from the entries of the storage, because after the increment the counter we write the current time in the user record in database. Comparing the current time and the time of last user increment the counter will be made by Lua-script, which we will connect to the Nginx by configuration file.

PROGRAMMING LANGUAGES

Traditional back-end programming languages are not need, but developer needs a nginx configurating skill and Lua-scripting skill.

For browser-side part of service you can use a JavaScript-code for use the Ajax technology to send POST HTTP about event of button click and/or authorization, besides you can use a JavaScript language (with jQuery if you want) to forming a hash of login and password, which sends to server with auth-request.

REST-API

1. Get counter's value.

- https://example.com

for getting the page with counter's value

- https://example.com/getvalue

to get the counter's value without HTML-page.

2. Authtorization (only if necessary).

POST request to https://example.com with the following parameter:

- Hash.

Hash save a value of hashing concatenation of user's login and password.

3. Increment.

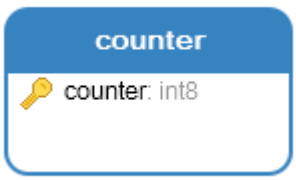
https://example.com/inc

For execute the operation with request must send a cookie, that must contain the authorization sequence or send a authorization hash if we are using a non-cookies way.

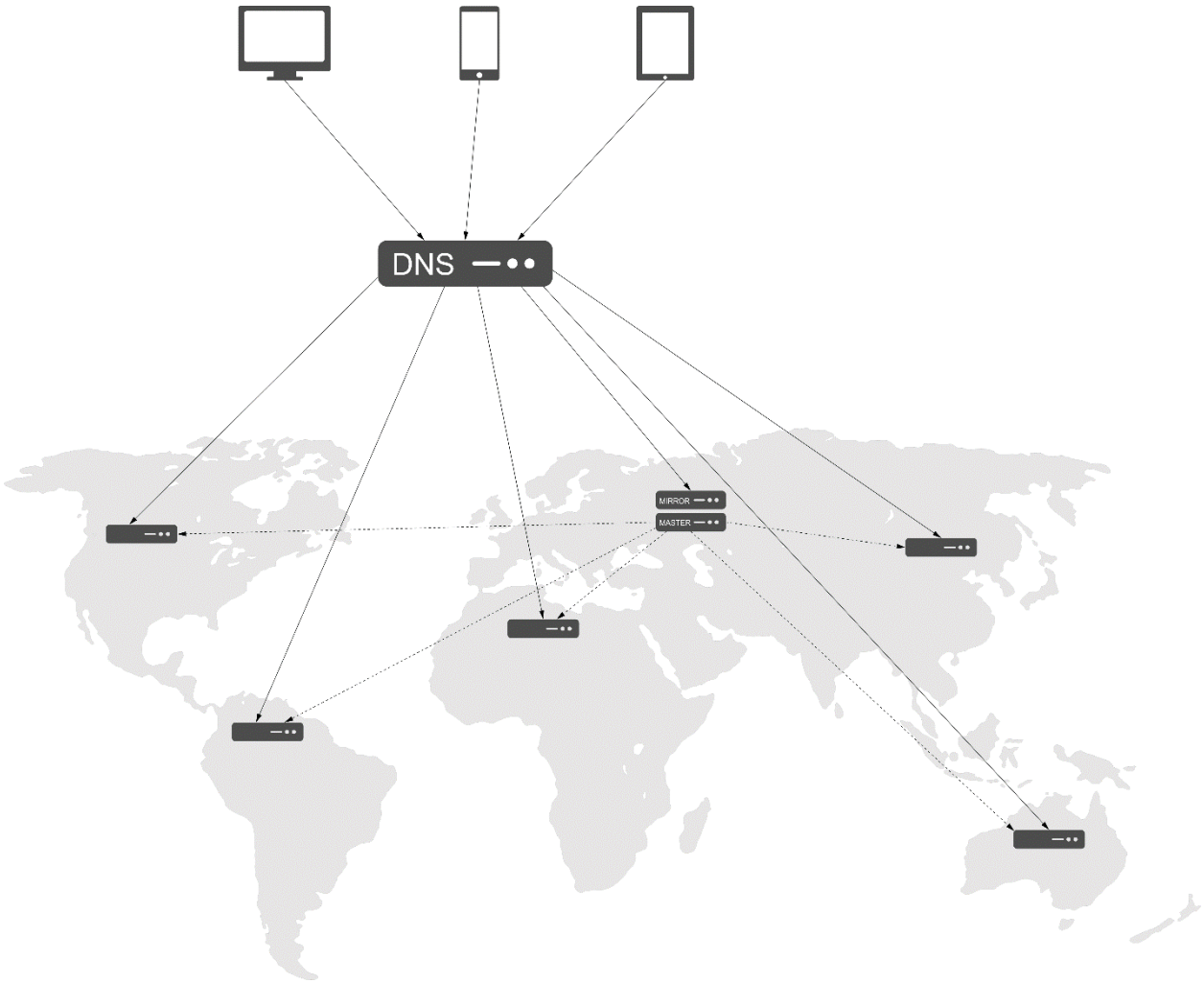
DATABASE SCHEMA

None of Redis.

For PostgreSQL - 1 database with a single table containing a single cell:



HIGH-LEVEL ARCHITECTURE DIAGRAM



The integrity of the data storage is guaranteed transactional nature of Redis / PostgreSQL.

FOR SAFETY

In all of the slaves is open only port 80 for connect to Nginx web-server by browser and a port for storage replication, for management the slaves administrator must to connect to it through the master (use the master as a proxy). In the master, open port 80, replication port and port of connect via SSH . Connecting via SSH is only available with a private key. When authorization, user hash from login and password is generated in a client browser and sent to the server, thereby preventing attacks such as Man-in-the-middle. For greater security we can disable authentication using cookie and ask the user to enter a username

and password each time when he tries to increase the counter value, as we mentioned above.