Обзор

Из определения задачи ясно что система должна быть доступна из любой точки земного шара, при этом желательно получать страницу с кнопкой и числом за условные 300 мс и менее (при отклике за 300 и менее миллисекунд у пользователя не создается ощущение что он ждал). Причем запросов на чтение предполагается гораздо больше чем запросов на запись.

Ввиду географической распределенности пользователей лучше всего использовать географически распределенный кластер серверов.

В задаче ничего не сказано о требованиях к актуальности данных. Если предположить что каждый пользователь не обязательно должен получать самое последнее значение счетчика, например, достаточно будет показывать ему значение счетчика, которое было актуально 2 секунды назад, то мы можем использовать кластер с мастером и его зеркалом в главной датацентре и 5-ю географически распределенными слейвами – по одному на каждом континенте, не считая континента на котором находится мастер с его зеркалом и Антарктиды. Мастер будет обрабатывать запросы на запись и реплицировать данные счетчика на другие сервера кластера – это самый производительный узел в кластере. Зеркало может быть слэвом по сути или же быть вторым мастером, так же принимающим запросы на запись. Таким образом, есть 2 пути его использования:

1. Основной мастер работает запись, зеркало работает только на чтение; репликация: мастер-слейв. В этом случае при падении мастера новым мастером становится именно зеркало. Данные, котрые не успели среплицироваться на зеркало могут быть утеряны, но могут быть и сохранены при использовании двунаправленной репликации (BDR).
   Упавший мастер не принимает запросов на время поднятия, все его клиенты начинают обращаться к зеркалу, сессии пользователей теряются.

   ПО в этом случае, без друнаправленной репликации:
   - для мастера и слевов: Debian + nginx with redis extension + redis;

   с двунаправленной репликацией:
   - для мастера: Debian + nginx with postgresql and redis extensions + postgresql + redis.
   - для слевов: Debian + nginx with postgresql extensions + postgresql

   PostgreSQL будет использоваться для хранения счетчика, и вводится по причине того что этот инструмент поддерживает двунарправленную репликацию, а Redis не поддерживает.

2. Оба узла работают на запись и чтение; репликация данных: мастер-мастер. Балансировка между ними: Sticky Sessions.

ПО (мастера и слейвы) : Debian + nginx with redis extension + redis,
Использование двунаправленной репликации может увеличить производительность, но тогда может оказаться так что ни на одном из меастеров никогда не окажется полностью актуальные данные – отказываемся от этого варианта.

Первый способ выглядит предпочтительным, так как дает прирост в производительности записи, надежность у обох подходов одинаково хорошая (при условии использование двунаправленно репликации хранилища данных в первом способе). В свою очередь использование в качестве основного хранилища Redis в первом подходе даст немного большую производительность и на запись и на чтение, но, к сожалению, допустит вероятность потери небольшого количества транзакций при падении мастера. В дельнейшем будем рассматривать оба варианта.

В случае падения мастера и его зеркала или отклчюение от сети основного ДЦ мастером должен стать слейв с одного из других континентов – первый по счету или весу.

Слейвы, как уже было сказано, будут распределены географически и будут обрабатывать только запросы на чтение, причем со своих континентов. Данные на них могут актуализироваться с репликационной задержкой, таким образом не всегда все пользователи будут видеть самое последнее значение счетчика.

При падении слева он временно отключается от обработки запросов.

При добавлении нового слейва он наполняется данными с мастера и запускается в работу посредством DNS.

Запросы на авторизацию, как и запросы на инкремент счетчика всегда обрабатываются только мастером (мастерами), на слевах будут хранится только данные о значении счетчика.

Роутинг запросов должен производиться на уровне DNS при помощи технологии Anycast.

Если же актуальность данных принципиальна нам придется использовать, либо распределенный мастер-мастер, либо не распределять узлы кластера географически – держим все в одном ЦОД и надеемся на некритичность задержки для пользователей из Австралии.

Стоит отметить, что проблемы распределения и репликации данных касаются только хранилища. ПО и конфигурационный файлы дублируются на всех машинах.

Касаемо hardware, основные условия:

1. Объединение дисков каждого узла в аппаратные RAID-массивы с батарейкой, например, с уровнем 10.
2. Желательно иметь на каждом узле такое количество оперативной памяти, чтобы туда вмещались все необходимые данные.

Архитектура узлов

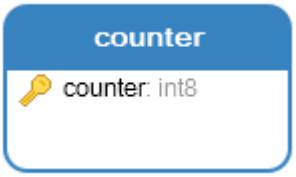
Запрос на чтение приходящий на узел обрабатывается при помощи Nginx в 4 этапа: получает разметку страницы из файла, получает значение счетчика из Redis/PostgreSQL, сводит их вместе и отдает клиенту. 2 первых этапа могут проходить параллельно, «сведение» значения счетчика и страницы-шаблона можно провести про помощи, например, SSI.

При регистрации пользователей в Redis по ключу со значение хэша от конкатенации логина, пароля пользователя и какой-либо соли пишется некое значение. Хэш от логина и пароля приходит с клинта, а соль добавляется позднее – при формировании запроса к Redis о сохранении записи о новом пользователе. По этому ключу-хэшу в Redis мы положим значение true и авторизуем пользователя при помощи cookie.

Дальнейшая авторизация происходит при условии нахождения вышеописанного ключа в хранилище Redis.

Есть и более безопасный вариант, когда мы запрашиваем логин и пароль только при попытке пользователя увеличить значение счетчика. Тогда хэш от логина и пароль приходят в месте с запросом на инкремент, мы проверяем пришедшие авторизационные данные и в случае их легитимности увеличиваем счетчик, при этот не устанавливаем cookie и при следующей попытке инкрементировать счетчик снова попросим пользователя ввести логин и пароль. Однако, этот способ при большей безопасности может быть не удобен с точки зрения user experince.

Если для счетчика используется база данных PostgreSQL, в ней придется создать одну таблицу с одним столбцом и одной строкой, в единственной получившейся ячейке и будет храниться значение счетчика. Выглядит достаточно странно, но это плата за использование BDR.



Если же счетчик хранится в Redis, то все проще – мы просто пишем значение в хранилище по ключу, например, *counter*.

Запись производится инкрементом счетчика в хранилище при этом она доступна только авторизованным пользоватлям - тем кто прислал легитимные cookie и тем кто обновлял счетчик более 24 часов назад, значение времени последнего инкремента от пользователя мы достанем из записи в хранилище, так как после инкремента счетчика по хэшу пользователя в базу данных мы запишем текущее время. Сравнение же времени предыдущего инкремента со временем текущей попытки будет производиться при помощи скрипта на языке Lua, который мы подключим в конфиге Nginx.

Языки программирования

Традиционные языки программирования не нужны, но разработчику понядобятся навыки конфигурирования Nginx и программирования на языке Lua.

Для клиентской части сервиса можно использовать скрипты на языке JavaScript для отправки Ajax-запросов методом POST в ответ на клик по кнопке увеличения счетчика и/или попытку авторизации. Кроме того при помощи JavaScript можно получать хэш от логина и пароля пользователя перед отправкой на сервер.

REST-API

1. Получить значение счетчика
   - https://example.com
     для получения значения вместе со страницей;
   - https://example.com/getvalue
     для получения только самого значения.

2. Авторизация (если нужно).
   POST запрос к https://example.com со следущим параметром:
   - Hash.

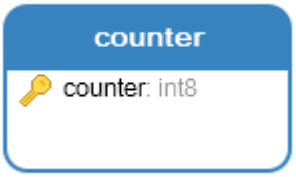
   Hash значие хэша от конкатенации логина и пароля введеных пользователем.

3. Инкремент счетчика.
   https://example.com/inc
   Вместе со запросом должен быть отправлен хэш от логина и пароля или cookie-последовательность для авторизации операции.

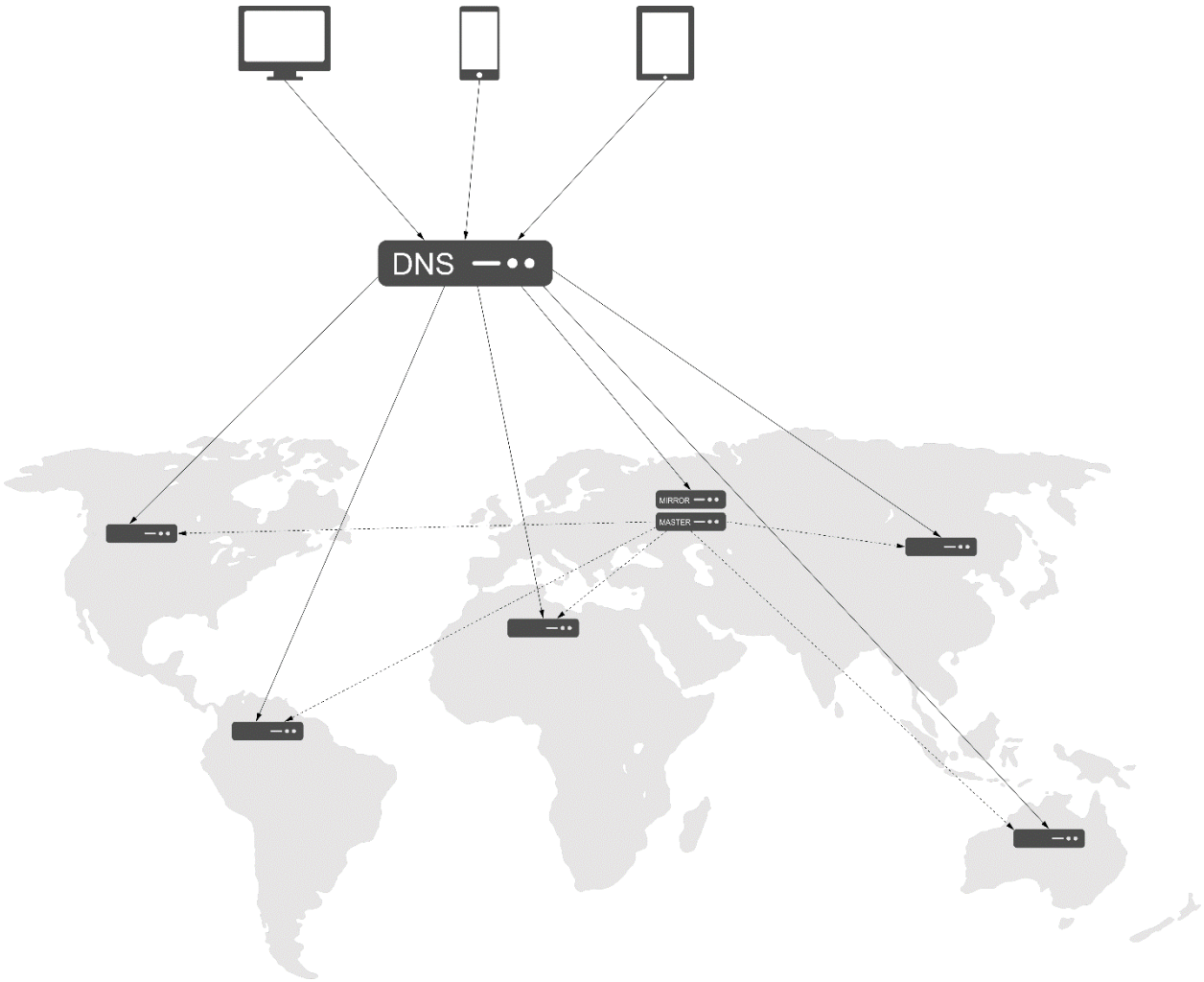
Схема базы данных

Отсутствует для Redis.

Для PostgreSQL – 1 база данных с одной таблицей, содержащей единственной значение:



Высокоуровненвая архитекрурная диаграмма



Целостность хранилища данных гарантируется транзакционной природой Redis / PostgreSQL.

Для безопасности

На всех слейвах открыт только 80 порт для поключения к nginx, для управления слевом нужно подключаться к нему через мастер (использую мастер как proxy) и порт для репликации хранилища. На мастере открыты 80 порт, порт для репликации и порт для подключения по SSH. Подключение по SSH доступно только с private key. При авторизации пользователей логином и паролем в клиентском браузере формируется хэш от логина и пароля, он и отправляется на сервер, таким образом исключается атака типа Man-in-the-middle. Для большей безопасности можно отключить авторизацию при

помощи cookie и просить пользователя вводить логин и пароль при каждой попытке увеличить счетчик.