СОЗДАНИЕ КЛАСТЕРНОГО СЕРВЕРА И ЕГО НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ

CREATING A CLUSTERED SERVER AND LOAD TESTING

R. Bondarenko
A. Romanenkov

Summary. Modern information services process a large number of requests from users. Their fast processing and stability are the two most important criteria that every modern, in-demand service should have. One of the best solutions for creating a stable server with high computing power is a cluster. A cluster is a system consisting of several computers connected to each other by a network, representing, from the user's point of view, a single hardware resource. To assess whether the computing power of the server is sufficient for the expected load, it is necessary to carry out load testing, through which to take care of the quality of the service provided.

Keywords: Web server, cluster, cluster server, load testing, information service.

Бондаренко Роман Николаевич
Национальный Исследовательский Университет Московский авиационный институт
rnikbond@gmail.com
Романенков Александр Михайлович
К.т.н., доцент, Национальный Исследовательский Университет Московский авиационный институт; с.н.с., ФГУ «Федеральный исследовательский центр «Информатика и управление» Российской академии наук»
romanaleks@gmail.com

Аннотация. Современные информационные сервисы обрабатывают большое количество запросов, поступающих от пользователей. Их быстрая обработка и стабильность работы — два самых главных критерия, которым должен обладать каждый современный, пользующийся спросом, сервис. Одним из наилучших решений для создания стабильного сервера с высокой вычислительной мощностью — кластер. Кластер — это система, состоящая из нескольких, связанных между собой сетью компьютеров, представляющая, с точки зрения пользователя, единый аппаратный ресурс. Для оценки, достаточно ли вычислительной мощности сервера для предполагаемой нагрузки, необходимо проводить нагрузочное тестирование, благодаря которому заранее позаботиться о качестве предоставляемого сервиса.

Ключевые слова: Веб-сервер, кластер, кластерный сервер, нагрузочное тестирование, информационный сервис.

Введение

Доступ к большинству современных сервисов осуществляется посредством глобальной сети интернет. Для возможности функционирования любого сервиса необходим сервер, которым является компьютер с установленными необходимыми инструментами. *Стабильность* и *быстродействие* являются одними из важнейших критериев любого современного сервиса [1]. Одним из наилучших решений, который гарантирует выполнение обоих критериев при его правильной организации, является *кластер*. *Кластер* — это система, состоящая из нескольких, связанных между собой сетью компьютеров, представляющая с точки зрения пользователя единый аппаратный ресурс [2]. Поскольку кластер является гибким к расширению, необходимо оценить его вероятную нагрузку, которая выражается в виде количества поступаемых запросов от пользователей в единицу времени. Используя *нагрузочное тестирование*, можно проверить, какое количество клиентов способен обработать сервер.

Архитектура кластера

Реализация сервера как кластера является наиболее удачным решением, поскольку этот способ позволяет решить следующие проблемы:

1. Сохранение работоспособности сервера после выхода из строя одного или нескольких составных компонентов.
2. Гибкое увеличение вычислительной мощности сервера.

Для дальнейшего понимания, на рисунке 1 представлена схематичная структура сервера в кластерной реализации.

В данной схеме показаны 3 основные составляющие группы *кластера*:

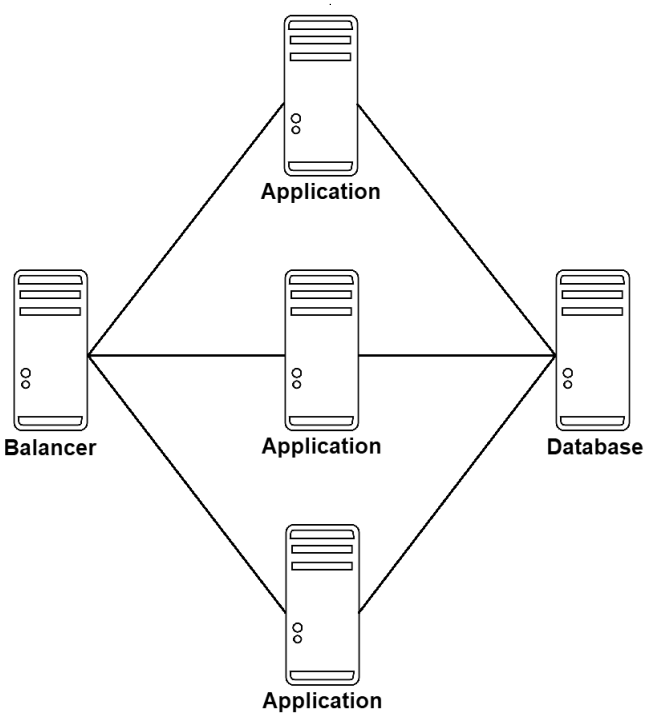

Рис. 1. Архитектура кластера

1. *Balancer* — сервер для распределения нагрузки.
2. *Database* — группа серверов с базами данных для хранения информации.
3. *Application* — группа серверов, на которых установлены программные экземпляры сервиса, занимающихся обработкой запросов.

Все запросы поступают на сервер *Balancer* (*балансировщик*), которой распределяет их между серверами группы *Application*, используя один из методов распределения нагрузки (*балансировки*). Сервера группы *Application* обрабатывают поступающие запросы и отправляют результат (ответ на запрос) обратно на *балансировщик*, который, в свою очередь, отправляет его клиенту, отправившему запрос.

Методы распределения нагрузки

Распределение нагрузки между серверами — это один из ключевых моментов, который в значительной степени влияет на быстродействие сервиса на программном уровне. Нельзя выбрать один метод *балансировки*, который одинаково хорошо подходил бы для любого сервиса и для любых компьютеров (в аппаратном смысле). В веб-сервере *nginx* реализовано несколько методов распределения нагрузки, наиболее эффективные из которых рассмотрены ниже.

1. *Round-robin* — алгоритм данного метода реализован так, что он распределяет нагрузку равномерно между всеми доступными серверами. Его стоит использовать тогда, когда все сервера имеют одинаковую вычислительную мощность. В случае же, когда одни сервера обрабатывают запросы быстрее, а другие медленнее, можно настроить *веса* (*weight*) [3], которые по умолчанию равны единице. Но это уже ручная настройка, которая требует дополнительного анализа для выбора оптимальных параметров *весов*. В таком случае лучше подойдет один из методов *Least-connected* или *Least-time*, речь о которых пойдет ниже. Поскольку этот метод (*Round-robin*) используется по умолчанию, директива для его включения отсутствует.
2. *Least-connected* — данный метод передает запросы на наиболее свободный сервер (тот сервер, у которого наименьшее количество активных соединений). В случае, когда сервера одинаково загружены, сервер для обработки запроса выбирается циклически, при помощи вышеописанного метода *Round-robin*. Для использования данного метода следует включить директиву *least_conn* в конфигурационный файл, пример которого представлен ниже:

```
upstream cluster {
least_conn;
server 10.0.2.6;
server 10.0.2.7;
}
```

3. *Least-time* — метод *балансировки*, суть которого заключается в выборе для обработки запроса того сервера, у которого наименьшая средняя задержка и наименьшее количество активных соединений. В качестве наименьшей средней задержки используются параметры: *header* — наименьшее среднее время получения заголовка ответа от сервера, *last_byte* — наименьшее среднее время получения полного ответа от сервера и *inflight* — время получения полного ответа от сервера с учетом незавершенных запросов[3]. Конфигурационный файл, в котором используется метод *Least-time*, представлен ниже:

```
upstream cluster {
least_time last_byte;
server 10.0.2.6;
server 10.0.2.7;
}
```

Реализация кластерного сервера

Представленная реализация производится на компьютерах с предустановленным дистрибутивом операционной системы *Linux*. Для сервера *Balancer* необходимо установить веб-сервер *nginx*, что можно сделать через терминал при помощи команды: *sudo apt-get install nginx*. После установки необходимо перейти в директорию */etc/nginx* и произвести настройку конфигурационных файлов. В директории */etc/nginx/conf.d* необходимо создать файл *upstream.conf* и заполнить его следующими конфигурационными данными:

```
upstream cluster {
    least_time last_byte; # Balancing method
server 10.0.2.6; # Application 1
server 10.0.2.7; # Application 2
    server 10.0.2.8; # Application 2
}
```

В этом файле указывается метод распределения нагрузки (*балансировки*) *Least-time*, наименьшая средняя задержка которого рассчитывается из среднего времени получения полного ответа от сервера. Далее происходит перечисление компьютеров — серверов группы *Application* с их статическими *IP адресами*. В случае необходимости можно добавить большее количество серверов.

Поскольку изначально запросы поступают именно на *балансировщик*, необходимо создать конфигурационный файл для получения данных от реализуемого сервера. В директории */etc/nginx/sites-available* создадим файл *cluster.conf* и запишем в него следующие конфигурационные данные:

```
server {
listen: 8000; # Port
server_name: cluster;
    access_log:  /var/log/nginx/cluster.access.
    log;
    error_log: /var/log/nginx/cluster.error.log;
    location / {
    proxy_read_timeout: 700;
    proxy_pass http://cluster;
    }
}
```

В строке *listen: 8000;* указан порт, который *прослушивает* веб-сервер *nginx*; *server_name: cluster;* — доменное имя, относящиеся к текущему виртуальному хосту. Далее указаны пути к файлам *логирования*, в которые *nginx* будет записывать как все текущие данные (*access*), так и данные об ошибках (*error*). В блоке *location / {...}* указана ссылка для доступа к сервису и время ожидания ответа на запрос. Теперь нужно активировать конфигурационный файл. В каталоге */etc/nginx/sites-enabled* необходимо создать символическую ссылку при помощи следующей команды: *cd /etc/nginx/sites-enabled/ && sudo ln -s /etc/nginx/sites-available/ cluster.conf*.

На этом шаге *балансировщик* настроен и сконфигурирован. Рассмотрим настройку одного из серверов, относящегося к группе *Application*. На всех остальных серверах этой группы настройка аналогичная, разница только в *IP адресе*. Также, как и на *балансировщике*, необходимо установить веб-сервер *nginx* и в директории */etc/nginx/sites-available* создать конфигурационный файл *cluster* и заполнить его следующими данными конфигурации:

```
server {
    server_name cluster;
    access_log   /var/log/nginx/cluster.access.
    log;
    error_log /var/log/nginx/cluster.error.log;
    root /var/www/cluster;
    location / {
        index index.php index.html;
    }
    location ~ \.php$ {
    try_files $uri = 404;
    include fastcgi_params;
    fastcgi_read_timeout 600;
    proxy_read_timeout 5;
    fastcgi_pass    unix:/run/php/php7.0-fpm.
    cluster.sock;
    fastcgi_index index.php;
    }
}
```

Конфигурационный файл сервера группы *Application* похож на конфигурационный файл сервера *Balancer*, но нужно обратить внимание на следующие параметры: *root /var/www/cluster* — здесь указан путь к самому приложению или, иначе говоря, программному экземпляру сервиса; *proxy_read_timeout 5* — данным параметром производится настройка сброса обработки запроса для случая, если он обрабатывается слишком долго. Здесь установлен интервал для обработки не более 5 секунд.

Каждый следующий сервер группы *Application* настраивается аналогично. В директории */var/www/cluster* находится программный экземпляр предоставляемого сервиса. На каждом сервере-обработчике должна находиться копия сервиса.

Для ускорения процесса настройки можно создать образ настроенной операционной системы и далее устанавливать его на каждый новый компьютер. Реализованный кластер гарантирует стабильность работы. При выходе из строя любой составляющей группы *Application*, сервис все также будет доступен, но в связи с выходом из строя одного из узлов, снизится и производительность до восстановления вышедшего из стоя сегмента кластера. Данные сервиса не будут утеряны, поскольку они хранятся в группе *Database*.

Пример нагрузочного тестирования

В качестве примера создадим сервис, который сортирует данные при помощи модифицированного алгоритма *быстрой сортировки*. Данный алгоритм хоть и был разработана более 40 лет назад, но все равно является наиболее широко применяемым и одним их самых эффективных алгоритмов. Основная идея — разбиение входных данных на два массива, используя опорный элемент. Но, поскольку данный метод сортировки существенно снижает производительность при малом количестве сортируемых элементов, использовалась модификация с применением алгоритма простых вставок. Реализация сортировки производилась по алгоритму, представленному в книге Д. Кнут «Искусство программирования» [7].

На компьютерах группы *Application* в директории */var/www* создадим папку *cluster* и в ней создадим файл *index.php*, исходный код которого в части алгоритма сортировки представлен ниже:

```
<?php
function QuickSort($input, $M) {
    if(count($input > $SZ) {
    push($left, 0); push($right, $SZ — 1);
    while($left->$size > 0) {
    // Q1 — Q8
$l = pop($left); $r = pop($right);
 $i = $l;$j = $r;
$pivot = $arrayStruct[($l + ($r — $l) / 2)].$number;
if($pivot > 1) {// Q2
    do {//Q3
        while($arrayStruct[i].$number < $pivot) {$i++;}//Q4
        while($arrayStruct[j].$number > $pivot) {$j—;}// Q5
            if($i <= $j) {// Q6
            if ($arrayStruct[i].$number > $arrayStruct[j].$number){
            $tmp = $arrayStruct[i].$number;
            $arrayStruct[i].$number = $arrayStruct[j].$number;
            $arrayStruct[j].$number = $tmp;
            }
            $i++;
            if ($j > 0) {$j—;}
    }
    } while ($i <= $j);
        if ($i < $r){// Q7
        push($left, $i); push($right, $r);
    }
    if ($j > $l){
        push($left, $l); push($right, $j);
    }
    if ($left->$size == 0 && $right->$size == 0) {//Q9
        for ($j = 1; $j < $SZ; $j++){//S2
            $i=$j—1;$k=$arrayStruct[i].$number;
            $r = $arrayStruct[j].$R;
            do{//S3
            if ($k >= $arrayStruct[i].$number) {break; //S4}
            else{$arrayStruct[$i + 1].$R = $arrayStruct[$i].$R;
            $i—;}} while ($i > 0);//S5
            $arrayStruct[$i + 1].$R = $arrayStruct[$i].$R;
    }}}}
    } else {//S1
        for ($j = 1; $j < $SZ; $j++){//S2
        $i = $j — 1;$k = $arrayStruct[$i].$number;
        $r = $arrayStruct[$j].$R;
        //S3
        do{if ($k >= $arrayStruct[$i].$number){//S4
        break;}else{$arrayStruct[$i + 1].$R = $arrayStruct[$i].$R;
    $i—;}
    } while ($i > 0);//S5
    $arrayStruct[$i + 1].$R = $arrayStruct[$i].$R;}}}?>
```

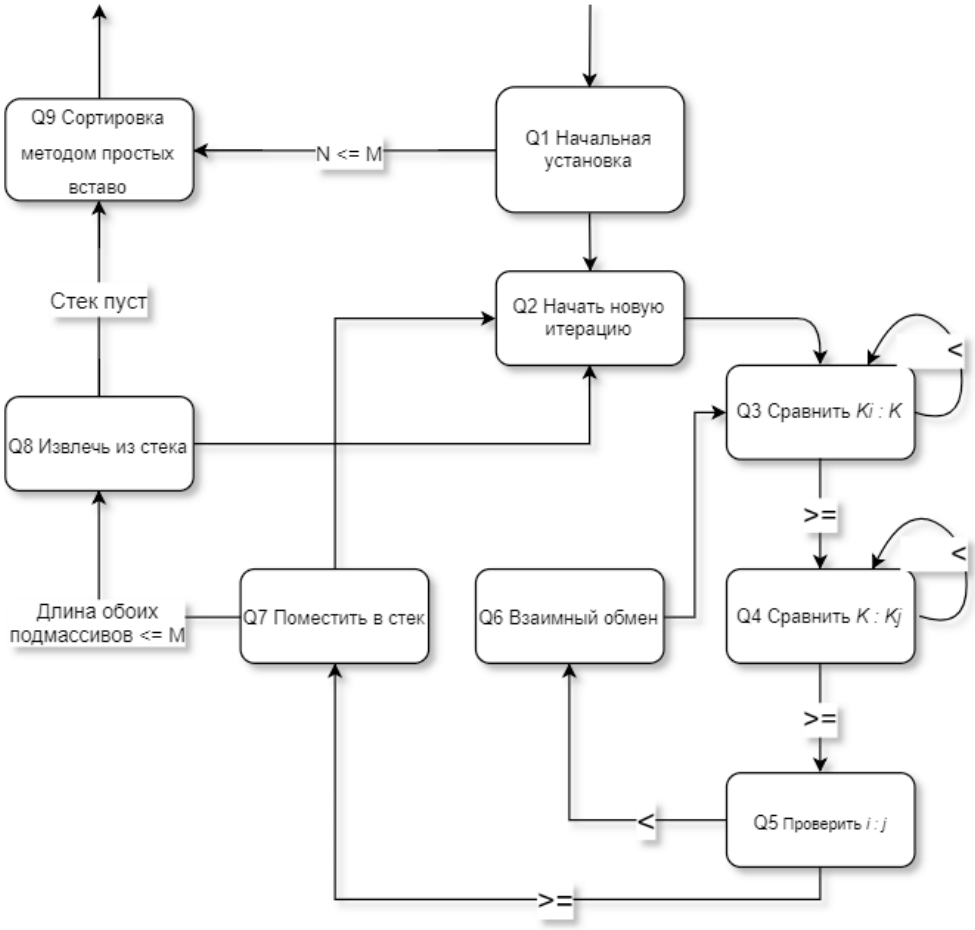

Рис. 2. Схема алгоритма быстрой сортировки

Схема реализованного алгоритма представлена на рисунке 2.

Сложность алгоритма зависит от входных данных и в лучшем случае будет равняться $O(N*2log2N)$. В худшем случае $O(N^2)$. Существует также среднее значение $O(N*log2N)$.

В данном сервисе в *HTTP* заголовке передается число N, соответствующее количеству элементов массива, после чего на серверной стороне создается массив из N случайно сгенерированных элементов.

Для нагрузочного тестирования используется утилита *ApacheBench* (далее *ab*). *ab* — это простой однопоточный инструмент командной строки для тестирования *HTTP*-серверов. Изначально он разрабатывался как часть *HTTP*-сервера *Apache*, но его можно использовать для тестирования любого *HTTP*- или *HTTPS*-сервера [4].

Базовый вызов утилиты *ab* выглядит следующим образом:

ab -n 10 -c 50 http://cluster:8000/index.php?num=30

Параметр *–n* задает количество запросов. Параметр *–c* задает конкурентность — имитация каждого пользователя. Затем указывается *URL*, который необходимо протестировать. Результатом является количество запросов в секунду, время запроса и список *процентилей* времени ответа, которое обработал сервер.

Для тестирования используется отдельный от кластера компьютер. После запуска нагрузочного тестирования при помощи утилиты ab, был получен результат, показанный на рисунке 3.

Тестирование производилось со следующими параметрами: 10 пользователей отправляют 30 параллельных запросов для сортировки массива из 30 элементов. Результат теста показал, что обработка всех запросов заняла *30.686* секунд, и в строке *Request per second: 1.63* сказано, что данный сервер способен обработать *1.63* запроса в секунду.

```
rnb conf.d # ab -c10 -n50 "http://cluster:8000/f.php?num=30"
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking cluster (be patient).....done


Server Software:        nginx/1.10.3
Server Hostname:        cluster
Server Port:            8000

Document Path:          /f.php?num=30
Document Length:        182 bytes

Concurrency Level:      10
Time taken for tests:   30.686 seconds
Complete requests:      50
Failed requests:        0
Non-2xx responses:      50
Total transferred:      17150 bytes
HTML transferred:       9100 bytes
Requests per second:    1.63 [#/sec] (mean)
Time per request:       6137.217 [ms] (mean)
Time per request:       613.722 [ms] (mean, across all concurrent requests)
Transfer rate:          0.55 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       1
Processing:  6110 6137  13.7   6143    6145
Waiting:     6109 6137  13.8   6143    6145
Total:       6110 6137  13.7   6144    6145

Percentage of the requests served within a certain time (ms)
  50%   6144
  66%   6144
  75%   6144
  80%   6145
  90%   6145
  95%   6145
  98%   6145
  99%   6145
 100%   6145 (longest request)
```

Рис. 3.Результат нагрузочного тестирования

Отключим два компьютера из группы *Application*, тем самым сымитируем их выход из строя, запустим снова тест и проанализируем файлы *логирования*, путь к которым был указан в конфигурационном файле при настройке балансировщика. Результат представлен ниже на рисунке 4.

Как видно из файла *cluster.access.log*, были получены коды *502*, *499* и *200*. Описание некоторых кодов возврата представлены в *таблице 1*.

В файле *логирования cluster.error.log* представлены записи, в которых описано, что сервера с *IP адресами 10.0.2.7* и *10.0.2.8* недоступны, что и соответствует действительности.

Заключение

Определив производительность, необходимо использовать эту информацию, чтобы попытаться улучшить время отклика сервера и снизить нагрузку на него. Благодаря нагрузочному тестированию есть возможность оптимизировать конфигурацию веб-сервера: изменить количество поддерживаемых подключений, рабочих процессов или потоков. Кластерный способ

Таблица 1. Коды возврата веб-сервера nginx

Код возврата	Значение
200	Запрос успешно обработан
499	Сервер разорвал соединение в процессе ожидания ответа
502	Сервер недоступен
504	Превышен интервал ожидания ответа

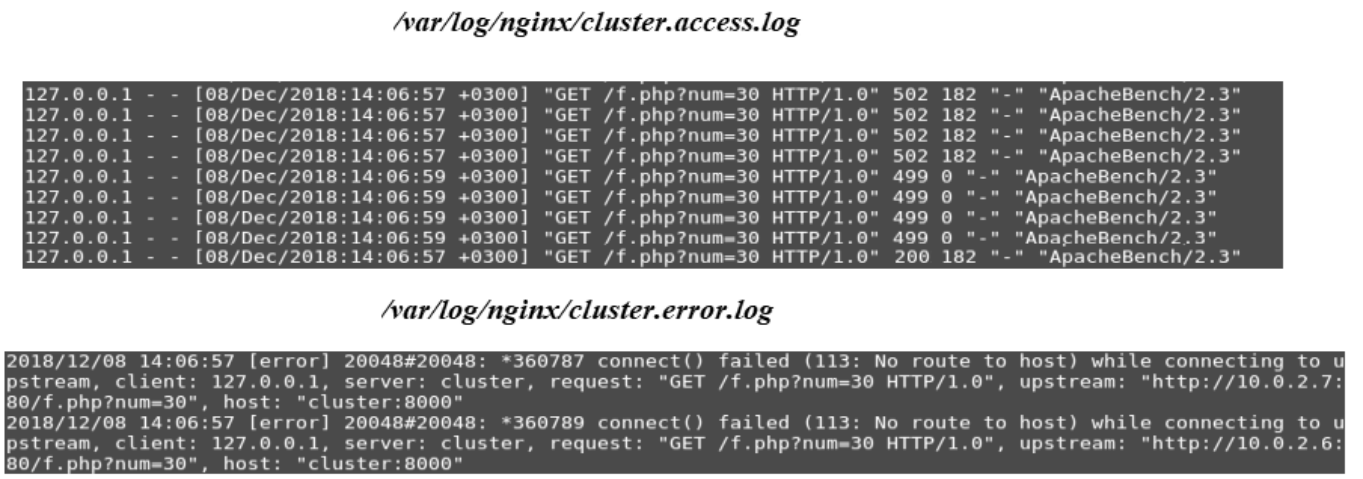

Рис. 4. Данные логирования при тестировании

организации сервера позволяет работать сервису стабильно за счет взаимозаменяемости элементов(компьютеров) в каждой группе. Также быстродействие достигается за счет распределения нагрузки при помощи наиболее оптимального метода балансировки. Результаты нагрузочного тестирования позволяют заранее позаботиться об улучшении сервера для поддержания сервиса в комфортном для пользователя состоянии.

ЛИТЕРАТУРА

1. Веб-сервисы в теории и на практике для начинающих: [Электронный ресурс]. 2008. URL: https://habr.com/ru/post/46374/ (Дата обращения: 01.03.2019).
2. Кластер (группа компьютеров): [Электронный ресурс]. 2018. URL: https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80. (Дата обращения: 10.03.2019).
3. Балансировка нагрузки HTTP: [Электронный ресурс]. 2018. URL: https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/ (Дата обращения: 10.04.2019).
4. Тестирование производительности веб серверов:[Электронный ресурс]. 2013–2015. URL: https://admins.su/site-speed-ab (Дата обращения: 06.03.2019).
5. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах. М.: Дело, 2007. 312 с.
6. Уильям, Р. Станек Internet Information Services (IIS) 7.0. Справочник администратора / Уильям Р. Станек. М.: Русская Редакция, БХВ-Петербург, 2012. 528 с.
7. Кнут, Д. Э. Искусство программирования. Т. 3. Сортировка и поиск. / Д. Э. Кнут. М.: Вильямс, 2014. 832 с.

Журнал «Современная наука: актуальные проблемы теории и практики»