Vizrt Community Expansion

Performance Guide

3.8.0.130433


vizrt

Disclaimer

Vizrt provides this publication “as is” without warranty of any kind, either expressed or implied.

This publication may contain technical inaccuracies or typographical errors. While every precaution has been taken in the preparation of this document to ensure that it contains accurate and up-to-date information, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained in this document.

Vizrt’s policy is one of continual development, so the content of this document is periodically subject to be modified without notice. These changes will be incorporated in new editions of the publication. Vizrt may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

Vizrt may have patents or pending patent applications covering subject matters in this document. The furnishing of this document does not give you any license to these patents.

Technical Support

For technical support and the latest news of upgrades, documentation, and related products, visit the Vizrt web site at www.vizrt.com.

Last Updated

13.09.2012


vizrt

Table of Contents


vizrt

1 Introduction

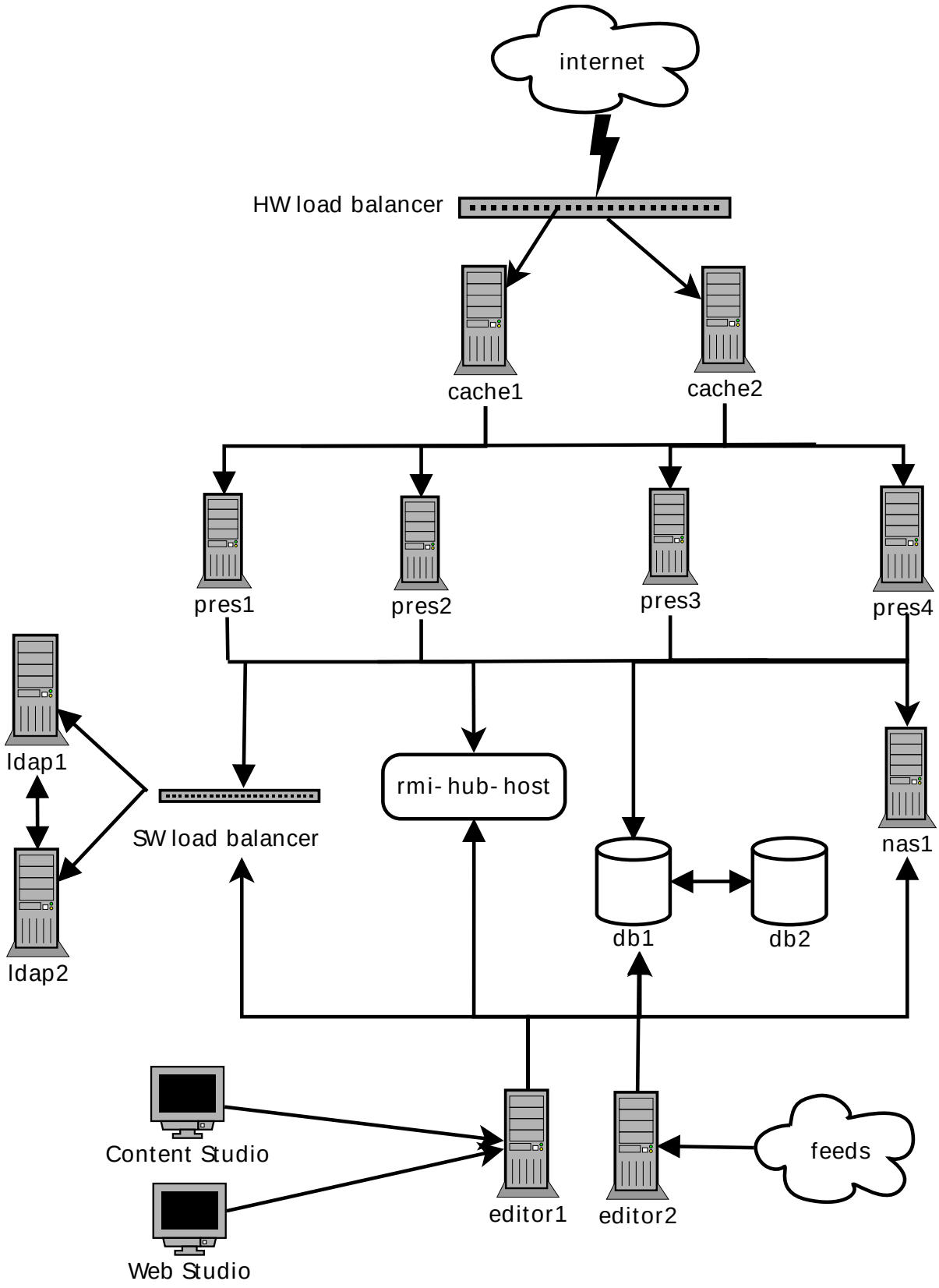
Welcome to the Vizrt Community Expansion Performance Guide. Getting a web site to perform well is a great challenge and adding thousands of logged in users, user generated content, personalisation and integration with 3rd party content to the mix only raises the bar.

Albeit a great challenge, getting all the server components to play nicely together is a great treat and ultimately gives the users of the site a better experience and ensures that the site has the best foundations for becoming a success.

This guide is meant as a starting point for identifying the challenges in this context and, even more importantly, solving them. It is assumed that the reader has a good understanding of the Escenic Content Engine, Unix (or GNU/Linux) server administration, networking, web servers, application servers, databases and cache servers. The guide is meant as a supplement to the Escenic Content Engine Install and Admin guides as well as the Vizrt Community Expansion Install Guide.

The guide is meant to be as general as possible, but where ever we discuss numbers (of servers etc), the text assumes you are building a community in the area of 50 000 simultaneous users.

The architecture in the following diagram should cater for such a number and has all the components discussed in the following chapters.



2 Getting a Normal ECE Site to Scale

To start with the easiest; getting a normal, high traffic ECE site to scale is "not a problem" as it's "only" a matter of caching the content right.

Roughly speaking, you need to make sure to have tuned the ECE caches (see the "Escenic Content Engine Admin Guide" for this), run a distributed memory cache (**`memcached`**) to ease the load on the databases (again, see the "Escenic Content Engine Admin Guide" for this) and a well configured cache server in front of the application servers, such as Akamai, Squid or Varnish.

The sizing of the server rig should be in the lines of 6-8 application servers, 2 DB servers, HA solutions (i.e. by using HA proxy and virtual IPs) for file system & RMI-hub and 2-4 cache servers (multiplied with two if you are using Squid 2.x).

3 Do Not Forget the Web Server

The cache servers will in most cases also run an Apache web server to make use of modules such as **mod_rewrite** and **mod_security** making the server indispensable for most web sites.

3.1 Web Server Tuning

The web server also needs tuning before going into production. The configuration that comes with the Apache distribution (or with our OS software package) is normally not optimised for high load web sites and thus, you need to tweak it. Especially the mpm_common worker module is important to configure for production use. Be sure to read and understand the documentation for this module and then continue to these more general Apache performance guides:

- http://httpd.apache.org/docs/2.2/misc/perf-tuning.html
- http://www.devside.net/articles/apache-performance-tuning

Be sure **not to use the prefork MPM worker**, but the (today normal) multi threaded worker.

The Apache worker is set during compile time. Thus, if you have compiled it from source, check your build (**configure**) options to be sure the multi threaded worker is selected. If you have installed Apache from RPM/DEB packages, you typically can use **rpm -qa | grep apache** or **dpkg -l "*apache*mpm"** to see make sure the high speed worker is being used.

This is an example of configuring the Apache worker for production use.

```
# worker MPM
<IfModule worker.c>
# We could increase ServerLimit to 64 and ThreadLimit/MaxClients to 8192,
# but be aware of the OOM of Death!!

# initial number of server processes to start
# http://httpd.apache.org/docs/2.2/mod/mpm_common.html#startservers
StartServers         3
ServerLimit          32

# minimum number of worker threads which are kept spare
# http://httpd.apache.org/docs/2.2/mod/mpm_common.html#minsparethreads
MinSpareThreads     512

# maximum number of worker threads which are kept spare
 http://httpd.apache.org/docs/2.2/mod/mpm_common.html#maxsparethreads
MaxSpareThreads     1024

# upper limit on the configurable number of threads per child process
# http://httpd.apache.org/docs/2.2/mod/mpm_common.html#threadlimit
ThreadLimit         4096

# maximum number of simultaneous client connections
# http://httpd.apache.org/docs/2.2/mod/mpm_common.html#maxclients
MaxClients         4096

# number of worker threads created by each child process
# http://httpd.apache.org/docs/2.2/mod/mpm_common.html#threadsperchild
ThreadsPerChild     128
```

```
# maximum number of requests a server process serves
# http://httpd.apache.org/docs/2.2/mod/mpm_common.html#maxrequestsperchild
MaxRequestsPerChild  10000
</IfModule>
```

Furthermore, we suggest that you have a good understanding of the following parameters. The values given here are suggestions that work well in production sites today. However, your needs may be different and you should therefore, as always, take heed when choosing (and sticking to) a number:

```
MaxKeepAliveRequests 1000
KeepAliveTimeout 5
```

3.2 Do Not Be Tempted

It might seem tempting to remove the web server to simplify the server setup. Especially since some cache servers (such as Varnish) has powerful URL rewriting, easy manipulation of HTTP headers and advanced access control lists.

However, encountering a site rig without a web server is rare and, if you plan to have personalised sites (with user login etc), session binding is a must. Some cache servers like Oracle Web Cache has this built in, whereas others, like Varnish does not (at the time of writing: 2010-02-08). Therefore, web servers are likely to be included in server setups for the foreseeable future. For more on session binding, see **section 7.1**.

4 Getting the Database to Scale

The real limitations of (mostly) read sites are the DB handles, as scaling the application server layer does not make sense when there is just "this many" read/write handles the database can handle. It might be that the higher end Oracle cluster solutions have something to offer here, but at least the MySQL cluster is not an option (it does not support sub queries), so you are stuck with a normal master/slave solution and this is what we believe **all** our current ECE sites are running with, regardless of the DB vendor.

It is important to remember that both the read and write connection pools in ECE **must be configured to work on the master DB instance**.

The slave DB(s) are only for data redundancy (standby backup) and should not be used to serve requests as this **may** cause unforeseen behaviour.

To ease the strain on the database, installing memcached on each ECE host is recommended. Memcached will act as a layer on top of the **`/neo/xredsys/presentation/cache/PresentationArticleCache`**, the most important ECE cache (called **`ArticlePresentationCache`** in ECE <= 5.1), and will reduce the database read operations significantly. Please see the "Escenic Content Engine Admin Guide" for details on how to install the memcached ECE integration.

5 The Good Old TCP/IP Stack

The next limitation to consider is the TCP/IP Stack. How many simultaneous open TCP connections can your front end servers handle, and what do these correspond to the backward (or downward if you like) TCP connections in your stack (application servers, DB, FS).

The TCP connections are not end to end, they are node to node specific. Hence connection handle wise, all layers of LB -> cache server -> app server -> DB/FS need to be in sync. For instance, there must be a sane relation between the number of connections made **to** the application server and **from** the application server **to** the database.

5.1 Caching Servers

For the top/front layer of your server architecture, i.e. your cache servers, it is also important to have a clear understanding of the scalability issues of TCP connections.

The first thing you may notice when putting load on your system, is that the cache server process runs out of file handles (if the start script does not increase the right kernel parameter). This is because the OS will use one file handle per connection (as this is what applications can relate to) and the default number of handles a given user process can create, (on many systems) is 1024. However, this is easily remedied with e.g. the **`ulimit -u`** command (on at least Linux and FreeBSD) or persistently using **`/etc/sysctl.conf`** (on Linux and FreeBSD) or **`/etc/system`** (on Solaris). The number of file handles can be cranked up to many hundred thousand and thus has no real limitation.

What is more interesting, is that the OS when creating the connection will create a connection from a local port to an anonymous port on the requesting host:

```
cache01:2323 -> ohterhost:1237
```

The port numbers are defined in the TCP protocol to be an unsigned 16bit number, yielding a maximum of 65535 ports available. Luckily, the local port number must not be unique across requesting hosts. Thus, the same port can be used for multiple connections to different IPs:

```
cache01:2323 -> ohterhost:1237
cache01:2323 -> yetanotherhost:4545
```

This means, that the maximum theoretical number of connections a cache server can handle, is **(65535 - <number of ports reserved for system services, normally 1024>) * incoming IPs**. For this to work as desired, it is important that the load balancer in front of the cache is fully transparent, exposing the incoming IP of the request to the cache server(s) and not the IP of the balancer itself.

To illustrate the last point, given the use case where three users are visiting your web site:

```
user1:2213 -> load-balancer:80 -> cache01:80
user2:1212 -> load-balancer:80 -> cache01:80
user3:5333 -> load-balancer:80 -> cache01:80
```

It is important that that **`cache01`** either sees the IPs of the requesting clients (**`user01`**, **`user02`** and **`user03`**) and not the IP of the load balancer, this is the optimal situation, or different IPs from of load balancer.

The latter solution is somewhat a hack, but it also works; just adding an additional interface/IP to the load balancer and/or the cache server will generate an additional set of origin host/port and destination host/port combinations, which will also make the cache server handle more than 65535 connections.

However, if you can, go with the first option and make the load balancer transparent. Your cache server will then be able to handle as many TCP connection as your load balancer can pass on (given that your OS kernel manages to allocate and recycle enough TCP connections fast enough, of course).

6 Searching with Solr

ECE and VCE make heavy use of the Solr search backend. The "Escenic Content Engine Install Guide" (for ECE >= 5.1) describes both normal setup of Solr and how to scale this component in a multi server environment and we advice the reader to follow this guide's recommendations.

7 Community Is Different

Ok, that was the easy part :-) Now, what makes the VCE sites different is that most of the content of the site is:

1. user generated
2. personalised
3. dynamic, i.e. the site shows different functions depending on if the user is logged in or not.

For these reasons, we must take more concepts and components into account. The following sections discuss the most important of these,

7.1 Session Binding

On an ECE site that uses profiles (so that users can be created on the page andthey can log in) and VCE sites, Apache is really helpful in providing sticky sessions and load balancing with its excellent **`mod_proxy_balancer`** module.

Please be advised that you cannot use the application server clustering (i.e. sharing a session between the application servers) as these, regardless of their backend or implementation, require that all objects written to the **`Session`** object are serializable. Currently, not all ECE objects fulfil this requirement and you therefore need to bind all sessions to a particular back end (i.e. application server). This binding can either be done in the web servers (as mentioned in the paragraph above) or on some cache servers, such as Oracle Web Cache.

7.2 ESI

> Put bluntly, having a clear ESI strategy in your template set is paramount to getting your community site to scale.

When dealing with the highly dynamic and user generated content nature of community sites, getting the dynamic bit to scale is "easily" done with ESI. ESI enables you to have different caching policy for different web page fragments and you can thereby improve site performance by differentiating between page content that updates often (e.g. the number of messages in a user's inbox) and less often (such as a news article or blog entry). Big IP, Varnish, Akamai, Oracle Webcache and Squid 3 all support ESI.

You must take heed, however, when developing your JSPs as using ESI means structuring your templates differently and setting the **`s-maxage`** HTTP header in entry point JSPs (the ones that receives HTTP requests directly and are not only included by other JSPs).

Because the application developer knows best how long a given fragment should be cached, he/she should tell the cache server how to cache the given fragment. At least with Varnish, you do not need any configuration to make the cache respect this directive in the JSP. This is an example on how to set the cache time of one minute on a fragment.

```
<%@ taglib uri="http://jakarta.apache.org/taglibs/response-1.0"
  prefix="response"%>
  <response:addHeader name="Cache-Control">
    s-maxage=60
</response:addHeader>
```

8 Single Point of Failure

To solve the problem of a single point of failure (which is what we inevitably have with the RMI hub and NFS), is to have the same software installed and configured on two hosts, where only is actually running it. A heart beat is running to check the availability of that service. In case it is down, the heart beat will start the service on the other (backup) host.

Included in the heart beat/fail over solution, should also be a virtual IP (often referred to as VIP) that the host running the critical software has. All servers that uses the service, relates only to the VIP. In case the host goes down and the heart beat starts the service on the backup host, the VIP is moved from the primary host to the backup host.

This way, the other servers will not have to change their configuration in any way. They will normally only lose their current transactions/connections (which were made to the server that went down) and operation will be resumed as normal with the consecutive requests/transactions, which then are handled by the backup host.

9 Optimising the Operating System Kernel

When installing an operating system, that system has settings that are so general that the OS can cater for a host of different scenarios. However, when you install a server to fulfill a certain task, you can tweak the OS kernel to optimise its behaviour to the piece of software installed on it.

Tweaking the different kernels are outside the scope of this guide, please see the appropriate parts of your operating system documentation. Here are some starting points:

OS	Starting Point
Linux	To optimise the Linux kernel, you may edit the **`/etc/sysctl.conf`** file, see the **`sysctl`** and **`sysctl.conf`** man pages. Use **`# sysctl -a`** to list all current setting. The setting names can be found by browsing the **`/proc/sys`** tree. For instance, the setting **`net.ipv6.route.max_size`** corresponds to **`/proc/sys/net/ipv6/route/max_size`**
FreeBSD	FreeBSD has also **`sysctl`** available. The OS has a great article online called Tuning Kernel Limits, which describes the various options.
Sun Solaris	Solaris uses **`/etc/system`** for setting kernel parameters. See the reference on docs.sun.com for the many options available.

Here is an example on how to tune the Linux kernel (tested on 2.6.24) for running the Apache web server and Varnish cache server. Albeit some of the settings may be redundant, this configuration is known to work and has a proven track record of serving several high traffic web sites:

```
net.core.rmem_max=16777216
net.core.wmem_max=16777216
net.ipv4.tcp_rmem=4096 87380 16777216
net.ipv4.tcp_wmem=4096 65536 16777216
net.ipv4.tcp_fin_timeout = 3
net.ipv4.tcp_tw_recycle = 0
net.core.netdev_max_backlog = 30000
net.ipv4.tcp_no_metrics_save=1
net.core.somaxconn = 262144
net.ipv4.tcp_syncookies = 0
net.ipv4.tcp_max_orphans = 262144
net.ipv4.tcp_max_syn_backlog = 262144
net.ipv4.tcp_synack_retries = 2
net.ipv4.tcp_syn_retries = 2
```

10 Run to the box office!

Another topic that needs to be heeded with a large scale community site, is registration. The current ECE/VCE does not have any queuing of registration requests, so if you expect 10 000 users to register within a timespan of 5-10 minutes, you need to create some kind of queueing of this.

11 How to Test

We must distinguish between three types of testing: smoke, functional and load testing. Some tools are good for one kind of testing, some are good for others. To ensure that your web site works well, you need to test all three.

11.1 Smoke Testing

A good first test, is to use **`wget`** to verify that the site is delivering the content (at all) and how fast it does this over time. While issuing this command, watch the OS resources (using e.g. **`top`**, **`vmstat`** and **`iotop`**) and the performance summary pages in the **`escenic-admin`** web application.

This command will download the given section pages with all its linked resources, such as images, style sheets and JavaScripts. It should always be run several times as you may hit the system in a too good or too bad point in time: when the caches are being filled up, when the connection to the database needs to be re-established or when the JVM is performing garbage collection. For these reasons, the command is performed inside a for loop that executes it ten times:

```
$ for el in {0..9}; do
  date; time wget -p --delete-after http://mysite.com/ -o /dev/null;
done
```

This is also a good way of filling up the front end caches after these have been flushed (for instance after a new deployment of your portal software).

11.2 Functional testing

We have had good experiences using JMeter for functional tests. With this tool, you can write scripts that allows you to simulate a lot of users signing in, writing blogs and so on.

The drawback of JMeter is that it is not "mean" enough and hence does not put enough strain on your server stack to verify/suggest it can sustain real, high volume traffic.

11.3 Load testing

In this context, we would like to suggest two tools: Siege and httperf.

Siege is multi threaded and will quickly reveal weaknesses in your stack with regards to read operations.

`httperf` is also good for providing load, but excels with its scripting support. This allows you to write session scripts (i.e. what a given users would issue of GET, PUT, POST and DELETE operations). Furthermore, it can replay your

Apache access logs, giving your tests real user traffic patterns as opposed to looping through a list of URLs sorted in alphabetical order.

Here is an example of running **`httperf`**. The call will create 1000 connections with 20 requests in each, establishing 100 connections per second. Please see **`man httperf`** for an detailed explanation of the parameters:

```
$ httperf\
  --hog \
  --server myserver.com \
  --num-conn 1000 \
  --ra 100 \
  --num-calls=20
```