Movable Type Operations Manual

Contributors: Byrne Reese

Version: 1.3

Table of Contents

System Architectures

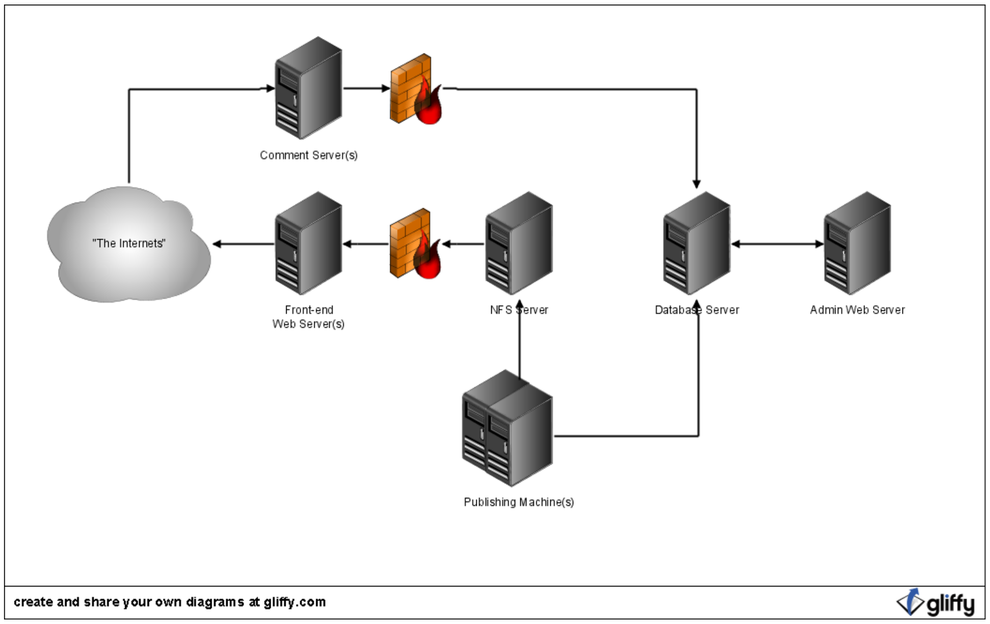
Introduction

Movable Type is renown in operations circles for a number of its "abilities," mainly its scalability, availability and reliability. The larger your site gets however, and the more readers you have creating content for you in the form of comments and posts, the more complex your network architecture may need to become. For many the process of architecting a scalable and high performant Movable Type system can be a daunting task, largely because the process is largely undocumented.

The truth is that there is no one, canonical way to design your Movable Type system or any system for that matter, which is most likely one of the primary reasons contributing to the lack of documentation. This guide therefore does not issue a specific recommendation on how to architect or design your Movable Type system. The intention of this document rather is to put forth a number of different types of system configurations which illustrates how certain problems are solved.

Large Footprint (Advanced)

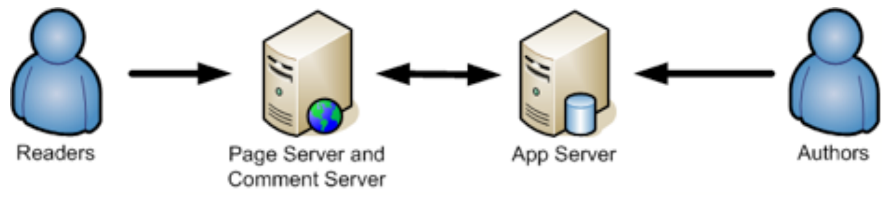
As a starting point lets dissect a basic, but large Movable Type footprint. While this is likely to be large for most users, understanding the logic behind such a layout will help you better understand the roles and functions a machine can have in a Movable Type environment.



- **Front-end Web Servers** - these servers serve static files only. If all else fails in your network these will continue to serve content, not to mention your ads, which is the life blood of any large media site.
- **NFS Server** - invest in a single, and very reliable, network storage device, one that your web servers read from, and your publishing daemons write to.
- **Database Server** - have one or two dedicated database servers depending upon whether you want one available for redundancy purposes. These should be beefy machines with a lot of CPU power and a lot of memory. There is a whole other chapter in fine tuning your database, and for that I highly recommend consulting an expert. In my experience however, text doesn't take up a lot of memory, and with a large enough cache configured for your database, you can practically load up your entire database into memory, resulting in dramatic speed improvements.
- **Comment Servers** - these web servers handle all the write requests from your community and readers including favoriting, commenting, and the like. These can be broken off from your front-end web servers so that they can be scaled independently from the rest. The diagram doesn't show this, but you may consider having these connected to the NFS server as well and have them handle publishing of your permalink pages synchronously with each comment received. This ensures that when a reader returns to the entry they commented on they will see the comment they just left.
- **Admin Web Servers** - these servers are what your editors access. It is given a dedicated machine so that if you site is under high load you can rest assured that authors and editors can still login and be productive administering the site.
- **Publishing Machines** - these servers are work horses. They handle much of the publishing and virtually all of the non-critical, non-blocking processes on your system, like Action Streams aggregation and most publishing. One simple way to approach this little cluster is with a lot of small cheap machines, or virtual machines that you can easily spin up when your site is under serious load.

Small Footprint (Common)

Above we outlined a large Movable Type configuration. For the vast majority of people, this setup will be overkill -- to put it mildly. So let's explore the other end of the spectrum[1]: the two server layout, a configuration that would be far more common.



TODO: better diagram

In this configuration work is shared between two servers: a front end web server and an app server. More specifically here are the functions each server performs:

[1] Technically, the "other end of the spectrum" is a single server layout, but it is not necessary to document that configuration given that it should be obvious: all software and all functions are performed on a single machine.

- **Front End Web Server, a.k.a. “Page/Comment Server”**
    - Serves all static HTML, PHP, CSS and Javascript files
    - Serves the MT front end CMS application
    - Services all commenting requests
    - Services all searches
- **App Server**
    - Hosts your database
    - Hosts your publishing workers

Setup and Configuration

CPUs, RAM, Disk Space and Resource Allocation

With two or more servers at your disposal the next logical question is how they should be configured? How much RAM should they have? How many CPUs? How much disk space? Etc. The answer to these questions are actually very simple: *provide as many resources on these machines as your budget permits*. With dedicated hardware, most hosting providers will provision more than adequate resources to each of your servers so this should not be a problem.

Dedicated Publishers: NFS vs. rsync

When you dedicate hardware to publishing in any way shape or form, then one must resolve how files published on one server will make their way to the public web servers. There are two ways to solve this problem: to copy the files using rsync, or to publish directly to a filesystem shared between your servers via NFS.

NFS

NFS, or Network File System, refers to a shared file system. One server can “export” a directory and make it mountable by another server. Another server then mounts that directory and can read from and write to it as if it was a native directory.

Here are the pro’s and cons of using NFS to assist in publishing across multiple servers in Movable Type:

Pros	Cons
Easy to setup. Faster to publish files across multiple front end servers. Scales much better for web farms with *lots* of front end web servers.	The shared filesystem is a single point of failure in your system. NFS can be slow between two machines that do not share a really fast, low latency network connection.

rsync

rsync is a simple protocol to help keep two disparate file systems completely synchronized. Movable Type can be configured to use rsync such that for each file that is published by Movable Type, that file will be transferred (via SCP or FTP or you-name-it) to one or more other servers.

Pros	Cons
Ideal for systems in which the publishing machine and the front end web server(s) do not share a low latency connection (e.g. Amazon S3 or EC2). When rsyncing, files are physically copied between servers so that front end web servers become redundant and thus offer greater reliability and availability for your content should one of your front end web servers be compromised.	A little more complex to setup. Can be slow if there is a need to rsync files to multiple (more than 3) front end web servers. While files are being synced, front end web servers can potentially be serving different content for a short period, in other words, updates are not instantaneous across front end web servers. When utilizing this method, all templates must be published via publish queue. You cannot mix static and background publishing.

Recommendation

Generally speaking, NFS is typically the most common selection among Movable Type users. It is selected for its ease of setup, its flexibility and because most of the time, servers within a cluster share a very fast connection between each other.

When choosing NFS, given that you are creating a single point of failure in your system, it is advisable to utilize highly reliable storage (e.g. RAID) for your published files.

Installing Movable Type on a CentOS System

Overview

This guide covers the procedures for setting up an entire server designed to run a Movable Type System. The server discussed will serve as the blueprint for the baseline system configuration.

What you will need

- A server with CentOS release 5.2 (Final) installed[2]
- Root access
- Internet connectivity

It is also assumed that your system already has yum installed, which should be a standard component of any CentOS system.

What will be done

Before the actual instructions are provided, let's first cover what we intend to install and ensure is operational before we continue.

1. Setup Apache
2. Install Fast CGI
3. Install MySQL
4. Install PHP
5. Setup directories for logging
6. Install all prerequisites for Movable Type
7. Install Movable Type and Motion
8. Setup the Movable Type run-periodic-tasks script
9. Install and setup Sphinx Search

[2] If you are unsure what linux distribution you are running, check the contents of the file "`/etc/redhat-release`"

Installation and Setup Instructions

Execute the following commands as necessary:

1. Setup users for MySQL. Typically this should not be necessary because by installing mysql via yum this should be done for you automatically. However in some systems, this has failed. Therefore, by doing this manually you ensure that it will not fail later. Run the following command:
   ```
   prompt> /usr/sbin/groupadd mysql
   prompt> /usr/sbin/useradd mysql -g mysql
   ```

2. Create the directory we will use for capturing all Movable Type related logging info:
   ```
   prompt> mkdir /var/log/mt/
   prompt> chown -R apache /var/log/mt
   prompt> chgrp -R apache /var/log/mt
   ```

3. Download and install Yum GPG keys:
   ```
   prompt> rpm --import http://dag.wieers.com/rpm/packages/RPM-GPG-KEY.dag.txt
   prompt> rpm --import http://centos.karan.org/RPM-GPG-KEY-karan.org.txt
   ```

4. Setup additional yum repositories needed for Fast CGI and Memcached. We will create a new file called /etc/yum.repos.d/mt-CentOS-Extras.repo. See Configuration Files > Yum Repositories.

5. Install all required libraries and applications via yum. For each of the following applications you will enter the following command: "`yum install <package name>`" where "<package name>" should be replaced with the application/library you want to install. Each package may have additional prerequisites, in which case those additional packages must also be installed. Here is a list of all the packages you will need to install via yum:
   1. `emacs`[3]
   2. `ImageMagick`
   3. `mysql*`[4]
   4. `php*`
   5. `memcached`
   6. `mod_fcgid`
   7. `openssl`
   8. `glib`
   9. `freetype`
   10. `libtiff`

[3] Unless of course you are a masochist and use vi. :)

[4] If you are installing the MySQL server on a different machine, then there is actually no need to run "yum install mysql*" which effectively installs all mysql components onto your local system, including obviously the server itself. So in a situation where you are running the database on a different server, all you need to install run is "yum install perl-DBD-mysql"

11. `libxml`
12. `libxml2`
13. `ImageMagick-perl`

6. Initialize the MySQL database server:
```
prompt> mysql_install_db
```

7. Start the MySQL database server:
```
prompt> /etc/init.d/mysql start
```

8. Setup your Apache Virtual Host directories.
```
prompt> mkdir -p /var/www/vhosts/www.somedomain.com/htdocs/
prompt> mkdir -p /var/www/vhosts/www.somedomain.com/cgi-bin/mt/
```

9. Setup your Apache configuration file. See “Configuration Files.” Placing it in this directory will cause Apache to load it automatically, assuming you have a standard install of Apache on your web server.
```
prompt> emacs /etc/httpd/conf.d/mt.conf
```

10. Install all necessary Perl Modules. For each of the following Perl modules, you will need to execute the simple command “`cpan <module name>`”. Some of the modules below have additional prerequisites, and some have a lot of them. Be sure to “follow” and install all necessary prerequisites. *Pay close attention to what is output to the console and make sure all the perl modules are installed properly.* If a module is not installed, resolve the issue prior to continuing. See “Installing Perl Modules.”
Install these modules:
    - Convert::PEM
    - Archive::Zip
    - Archive::Tar
    - Crypt::DSA
    - XML::Atom
    - Mail::Sendmail
    - IPC::Run
    - FCGI
    - Cache::Memcached
    - Cache::Memcached::Fast
    - GD
    - XML::XPath
    - JSON::XS

11. Download Movable Type into your home directory.

12. Install Movable Type. You will need to download the latest version of Movable Type from Six Apart or from your customer account, and then copy the files into the proper location.
```
prompt> unzip MT-4.x.zip
prompt> cp -a MT-4.x/* /var/www/vhosts/www.somedomain.com/cgi-bin/mt/
```

13. Setup your Fast CGI scripts, which are nothing more than copies of the core Movable Type .cgi scripts using the file extension .fcgi.
```
prompt> cd /var/www/vhosts/www.somedoman.com/cgi-bin/mt
prompt> for file in `ls *.cgi`; do base=`basename $file .cgi`; cp $base.cgi
$base.fcgi; done
```

Note: The version of the Sphinx Search plugin currently available for download with Movable Type is non-functional. The following instructions will not work until such time as Six Apart updates their plugin.

14. Download and install the Sphinx search engine. Visit the following URL to download the Sphinx Search engine:
http://www.sphinxsearch.com/downloads.html
Then install it using the following sequence:
```
prompt> tar zxvf sphinx-0.9.8.1.tar.gz
prompt> cd sphinx-0.9.8.1
prompt> ./configure
prompt> make
prompt> make install
```
We will assume for the remainder of this manual that your sphinx.conf file has been placed in:
/etc/sphinx/sphinx.conf

15. Download and install Sphinx Search Plugin for Movable Type. Visit the following URL to download the Sphinx Search plugin for Movable Type:
http://www.apperceptive.com/plugins/sphinxsearch/
Then install it using the following sequence:
```
prompt> unzip SphinxSearch-0.99.8mt4.zip
prompt> cp -a SphinxSearch-0.99.8mt4/plugins/SphinxSearch \
    /var/www/vhosts/www.somedomain.com/cgi-bin/mt/plugins/
```

16. Set up the Sphinx Search Plugin for Movable Type by reading and following the instructions at this URL:
http://www.apperceptive.com/plugins/sphinxsearch/documentation.html
The instructions will guide you through the process of generating a sphinx.conf file and starting the searchd daemon.

17. Setup run-periodic-tasks by editing the crontab for the apache user:
```
prompt> sudo -u apache crontab -e -u apache
```
When your text editor comes up, you will need to enter the following into the file (all on a single line), save and then exit:
```
0,5,10,15,20,25,30,35,40,45,50,55 * * * * \
        cd /var/www/vhosts/www.somedomain.com/cgi-bin/mt; \
        perl ./tools/run-periodic-tasks
```

THE FOLLOWING DOES NOT WORK YET. DO NOT DO. Sorry.

18. Optional: Setup Sphinx init.d scripts to ensure that sphinx will start automatically if the server is rebooted. Visit the following URL and follow the instructions under “Create Searchd init.d Script:”
    http://www.notsofaqs.com/catsdoc/doku.php?id=sphinx:install
    Note: the /etc/init.d/searchd script provided will need to be edited to point at your sphinx.conf file. Look for:
    ```
    startproc -l $LOGFILE $SEARCHD --config \
         /srv/www/htdocs/cats/modules/search/sphinx.conf
    ```
    And change it to:
    ```
    startproc -l $LOGFILE $SEARCHD --config /etc/sphinx/sphinx.conf
    ```

Installing Perl Modules

Normally the process of installing Perl modules is very, very straightforward. Most systems come standard these days with a Perl package manager, not dissimilar to yum or apt, called "cpan." In almost all circumstances the process of installing a Perl module is as simple as running the following command as root:

```
prompt> cpan Perl::Module
```

What do to when things fail

Sometimes a Perl module will have tons of prerequisites, and for whatever reason, one of those prerequisites will stubbornly not install. The first thing to try is identify the perl module having difficulty, and try to install that module separately. Let's take a look at an example:

```
prompt> cpan Perl::Module
***** snip *****
---- Unsatisfied dependencies detected during [B/BY/BYRNE/Perl-Module-0.01.tar.gz] -----
XML::Simple
Shall I follow them and prepend them to the queue
of modules we are processing right now? [yes]
***** snip *****
Running make for G/GR/GRANTM/XML-Simple-2.16.tar.gz
Running make test
Make had some problems, won't test
Running make install
Make had some problems, won't install
```

Depending upon how many prerequisites are queued up for installation the key phrase "Make has some problems, won't install" may scroll by on your console too quickly for you to see. If you see a bad exit status when your prompt returns, however, scroll back up and identify which module had difficulty. In this case, it is XML::Simple. So now, try to install that module separately like so:

```
prompt> cpan XML::Simple
```

Doing this has two advantages:

1. It will help you to isolate the build error that occurred.

2. It may actually work. Why? I don't know, but sometimes CPAN has difficulty installing lots of Perl modules at once.

If this works and the module is installed properly, then try re-installing the Perl module you were trying to install initially.

If it *does not* work, then there is most likely a missing dependency or pre-requisite. Consult the documentation for the perl module that is failing and see if anything can be learned.

Under virtually no circumstances should you "force" the installation of a Perl module. If a unit test fails, then that is an indication that something is wrong. And if something is "wrong" it is best to determine what it is, resolve it, and then continue. This is a production system after all, don't you want everything to be perfect?

About the Publish Queue

The Movable Type Publish Queue is an essential component to any large scale Movable Type powered web site because it plays a crucial role in publishing performance optimization. There are a number of benefits to using the publish queue, they are:

- It **eliminates redundant, duplicated and unnecessary publication** of files.
- It **offloads publishing to stand alone process** which can be throttled and scaled independently from the Movable Type web application itself.
- It **speeds up the commenting experience** by reducing the number of files that an end user must wait to be published prior to being able to navigate the web site again.

How it Works

It might be best to describe how the publish queue works by examining a scenario in which it would be utilized: republishing the necessary files in response to a comment.

Adding Jobs to the Queue
When a comment comes in to Movable Type multiple files are often in need of being updated, not only because the comment needs to be published to the entry's permalink page, but also because multiple other pages which display a comment count associated with the comment's entry may need to be updated.

Each of those pages (assuming they are configured to be published via the publish queue) will then be added to the "publish queue." When this happens, a publishing "job" is created and added to the database for each page that need to be published. There is one row in the database for each individual job in the system.

Now let's assume for a moment that shortly after receiving the first comment, a second one is published by a different visitor to your web site. This action also results in pages needing to be republished. However this time, before those pages are added to the queue as jobs the system checks to see if a job corresponding to each page is already on the queue. If there is, then the job is discarded because its work would be unnecessarily duplicated otherwise. If the job is not already on the queue, then it is added. This ensures that no unnecessary work is performed by the system.

In addition, each page that is added to the publish queue is given a priority which dictates the order in which the corresponding job will be processed. The higher the priority, the sooner the system will work on the job. Movable Type assigns priority based upon the following criteria:

Page/Template Type	Priority
Preferred Page and Entry archives	10
Index templates with a filename beginning with “index” or “default”	9
Feed index templates	9
All other index templates	8
Non-preferred Page and Entry archives	5
Daily archives	4
Weekly archives	3
Monthly archives	2
Any Category archive	1
Any Author archive	1
Yearly archives	1

And that is how jobs are added to the queue. There is a separate process that exists that is then responsible for publishing.

Creating Publish Queue Workers
One or more publish queue “workers” can be created to process jobs on the queue. The number of workers needed by a system is based largely upon two variables:

- The capacity of any one worker to process jobs on the queue.
- The volume of jobs being added to the queue over time.

A worker is created by running the “run-periodic-tasks” script that comes with every copy of Movable Type. This script can be run in three modes:

- **daemon mode** - in this mode the script never quits; instead it constantly monitors the job queue for work to be done and nearly the instance a job is made available for work, the script will begin work on it.
- **run-once** - in this mode the script is run via the command line and will quit only after there is no more work on the queue to be done.
- **scheduled task** - in this mode the script is executed in the “run-once” mode periodically according to a schedule defined by cron or a similar service.

Processing Jobs on the Queue
Each worker will monitor the queue for jobs. When one becomes available it is pulled off the queue to be worked on. Once it is "off the queue" no other workers can claim it. This makes sure that no two workers are trying to work on the same job at the same time.

In the event that something goes wrong during the publishing process and the file is not published, then the system will notice saying something skin to, "uh-oh, look at this job that was claimed on the queue, but was never successfully finished," and then free up the job for a worker to pick up and try again on. If the task is retried more than 5 times, then the job is marked as failed and left on the queue. In this state it is possible for a similar job to be placed on the queue, and if the problem that was resulting in the published failure is not transient, then that job is likely to fail again.

An important thing to note is that if a job is pulled off the queue by a worker to be worked on, then it remains possible at that point in time for that same page to be added to the queue again in response to the receipt of another comment. The rational being that by the time the page is finished being rebuilt it is most likely out of date, and so needs to be published again.

What Powers It?

The Publish Queue is powered by a stand alone job/queue management library called “The Schwartz.” The Schwartz is actually a more generic and abstract job management system capable of processing any number of tasks via a similar queuing mechanism.

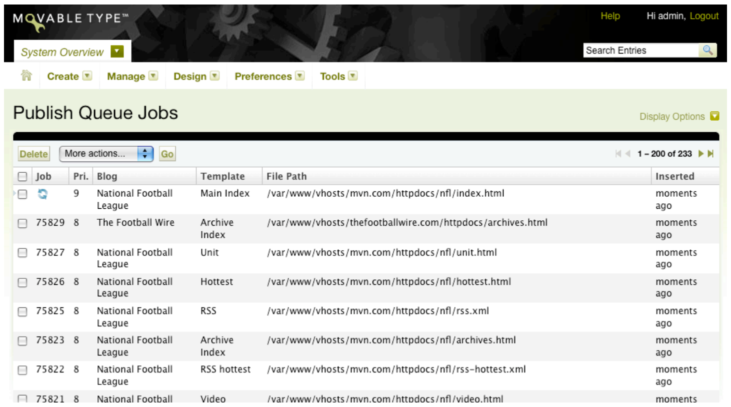
For the time being, Movable Type only utilizes the Schwartz for publishing, but in the future may use this framework for sending emails or other non-critical system tasks.

Publish Queue Tools

There is one tool in particular that is recommended for most systems that utilize the Publish Queue, aptly named the Publish Queue Manager.

This tool provides a user interface within Movable Type that allows administrators to monitor and inspect jobs on the queue. Each job can be deleted, or have its priority changed.

For more information, visit the plugin’s web site at the following URL:



Publish Queue Jobs

Job	Pri.	Blog	Template	File Path	Inserted
	9	National Football League	Main Index	/var/www/vhosts/mvn.com/httpdocs/nfl/index.html	moments ago
75829	8	The Football Wire	Archive Index	/var/www/vhosts/thefootballwire.com/httpdocs/archives.html	moments ago
75827	8	National Football League	Unit	/var/www/vhosts/mvn.com/httpdocs/nfl/unit.html	moments ago
75826	8	National Football League	Hottest	/var/www/vhosts/mvn.com/httpdocs/nfl/hottest.html	moments ago
75825	8	National Football League	RSS	/var/www/vhosts/mvn.com/httpdocs/nfl/rss.xml	moments ago
75823	8	National Football League	Archive Index	/var/www/vhosts/mvn.com/httpdocs/nfl/archives.html	moments ago
75822	8	National Football League	RSS hottest	/var/www/vhosts/mvn.com/httpdocs/nfl/rss-hottest.xml	moments ago
75821	8	National Football	Video	/var/www/vhosts/mvn.com/httpdocs/nfl/video.html	moments

http://www.majordojo.com/projects/movable-type/publish-queue-manager/

Using RSync and/or NFS in Multi-Server Environments

In large multi-server environments it often becomes necessary to take a single file that has been published by a Publish Queue worker and somehow get it to show up on multiple front end web servers.

Confused? Well, consider the following scenario. Suppose your system employs multiple machines for the express purpose of process publishing jobs on the publish queue. Now let’s say one of those machines updates one set of files and another one of those machines updates a different set of files. How then does one get these disparate files to a machine intended to serve them to your readers.

In a single server environment this is never a problem because the machine serving the files and the machine publishing them are one in the same. Therefore publish queue workers publish directly to your web servers document root and thus make updated pages available. In a multi server environment there are generally two different ways to solve this problem:

- Link your front end web servers and your publish queue machine together via a shared filesystem like NFS.

- Physically copy files from your publishing machines to your front end web servers via `rsync` or `scp`.

Now, let's explain each of these options in more detail.

NFS

In using the NFS solution all of your publishing servers (or Publish Queue workers) write files to an external NFS mount. In so doing these files never actually physically reside on the publishing server, they only appear to be local thanks to NFS which helps different servers share the same set of files between them. The front end web server then mounts this shared NFS directory for reading.

Pros	Cons
Scales better because each file is written once and immediately made visible on the front end web server.	Very poor performance in a geographical disparate setup (e.g. Amazon EC2 or other cloud services).
Easier to setup IMHO.	Single point of failure. If something were go wrong with your shared filesystem, then much of your system will be hosted. This can be mitigated with a solid RAID config or other highly reliable disk storage.

Note: "NFS" here is used only for illustrative purposes. Technically any shared file system technology will do.

RSync

When using `rsync`, Movable Type will invoke a command line utility designed for keeping two different file systems in sync with one another. This is what happens when Movable Type is configured to use RSync:

1. User leaves a comment.
2. Job is created in Publish Queue.
3. Worker pulls job off queue and publishes file to local file system.
4. Worker then begins to `rsync` (usually via `scp`) to each of the designated servers.

Pros	Cons
Failure tolerance - by replicating your published content you ensure that if one file system or server goes bad, you still have something to fall back on.	Slightly harder to setup IMHO.
Great for cloud hosting services like Amazon EC2, or any time in which your publishing server and front end web servers are not likely to be on the same subnet.	Scalability - the more front end web servers you have the more servers you will need to synchronize with. This can add latency to your publishing process and cause some servers for a brief period of time to have slightly different content from one another.
	Only works in Unix environments.

Setting Up Publish Queue and Rsync

To get started using Publish Queue and rsync you will need to follow these steps:

1. Make sure that your publishing servers are configured to publish files to the *exact same path* as your front end web servers are configured to read from. In other words, your publishing server should mirror exactly the file/directory/path structure of your front end web server.

2. Setup a user on your front end web server has that has write access to the directory that serves your blog's published files to the outside world. Make sure this user can connect via SSH to your front end web server from each of your publishing servers - *without having to supply a password.* This often done using SSH's special file called `.authorized_keys`.

Testing Your Setup

Once this is complete it is best to test make sure you can transfer files between the two hosts. To do so, successfully execute the following command from one of your publishing servers:

```
prompt> cd /
prompt> scp /path/to/a/file.txt username@someserver.com:/path/to/a/
```

If it is not obvious, please make sure to replace "/path/to/a/file.txt" with an absolute path to a file in your blog's document root. Also, replace "username" and "someserver.com" with the username and server address to transfer files to.

Your mt-config.cgi file

Once you have tested that files can be transferred between hosts *without being prompted for a password*, then add this to the mt-config.cgi file on each of your publish queue servers:

```
SyncTarget username@someserver.com:/
RsyncOptions -e ssh
```

Additional Reading

To learn more about the Publish Queue, consider reading the following resources:

- Using the Publish Queue
  http://www.movabletype.org/documentation/administrator/publishing/publish-queue.html

- Setting up run-periodic-tasks
  http://www.movabletype.org/documentation/administrator/setting-up-run-periodic-taskspl.html

- Scalable Publishing Models in Movable Type
  http://www.movabletype.org/documentation/enterprise/publish-queue.html

- The Schwartz Homepage
  http://search.cpan.org/~bradfitz/TheSchwartz-1.07/lib/TheSchwartz.pm

Performing a Server Migration

Preparing for the Move

Eset eiusmod tempor incidunt et labore et dolore magna aliquam. Ut enim ad minim veniam, quis nostrud exerc. Irure dolor in reprehend incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation.

Lorem ipsum dolor sit amet, ligula
Suspendisse nulla pretium, rhoncus tempor placerat fermentum, enim integer ad vestibulum volutpat. Nisl rhoncus turpis est, vel elit, congue wisi enim nunc ultricies sit, magna tincidunt. Maecenas aliquam maecenas ligula nostra, accumsan.

- One week ahead of time:
    - DNS prep - is the DNS TTL set to a low number?
    - notify readers of upcoming maintenence event
- Setup new machines
- Staging
    - Copy MT software to new environment
    - Copy Static Files and all published files to new environment
    - check file ownership and permissions of static files
    - export MT DB (mysqldump)
    - import MT DB in new environment
    - make necessary changes to mt-config.cgi
    - access MT
    - check publishing paths of all blogs
    - do a test publish of an entire blog
    - test creating/publishing an entry
    - test posting a comment
    - if everything checked out, then the environment is ready
    - save a copy of the mt-config.cgi file
- Migration Time

    - inform bloggers/authors NOT to publish new entries
    - perform mysql_dump
    - turn off trackbacks and comments globally
    - publish notice about maintenence event
    - shut down DB
    - perform mysql_dump
    - turn DB back on
    - import mt db data into new DB
    - start new DB
    - copy over static files
    - check file ownership and permissions of static files
    - copy the mt-config.cgi file you saved earlier over the one from the old environment
    - change DNS on your local machine
    - access MT
    - check publishing paths of all blogs
    - do a test blog post
    - do a test comment
    - if everything checks out:
        - delete local DNS change
        - change DNS globally
- In the event of a rollback:
    - login to the old system and turn comments back on
    - prepare to do it all over again

Appendix A: Configuration Files

The following section contains all of the various configuration files you will need:

Yum Repositories

- The following file should be placed in /etc/yum.repos.d/mt-CentOS-Extras.repo

```
# All new packages are now released to the testing repository first
# and only moved into Stable after a period of time
# Note: The testing repository is disabled by default

[dag]
name=Dag RPM Repository for Red Hat Enterprise Linux
baseurl=http://apt.sw.be/redhat/el$releasever/en/$basearch/dag
gpgcheck=1
enabled=1

[kbs-CentOS-Extras]
name=CentOS.Karan.Org-EL$releasever - Stable
gpgcheck=1
gpgkey=http://centos.karan.org/RPM-GPG-KEY-karan.org.txt
enabled=1
baseurl=http://centos.karan.org/el$releasever/extras/stable/$basearch/RPMS/

[kbs-CentOS-Testing]
name=CentOS.Karan.Org-EL$releasever - Testing
gpgcheck=1
gpgkey=http://centos.karan.org/RPM-GPG-KEY-karan.org.txt
enabled=1
baseurl=http://centos.karan.org/el$releasever/extras/testing/$basearch/RPMS/
```

Apache Configuration File

- The following file should be placed in /etc/httpd/conf.d/mt.conf
- Replace “127.0.0.1” with the IP address of your server

- Replace www.somedomain.com with the domain name of your server.
- Replace yourname@somedomain.com with your email address.

```
<VirtualHost 127.0.0.1:80>
  ServerName www.somedomain.com
  DocumentRoot /var/www/vhosts/www.somedomain.com/htdocs/
  ServerAdmin yourname@somedomain.com
  DirectoryIndex index.php index.html
  CustomLog /var/log/mt/access_log common
  ErrorLog /var/log/mt/error_log
  LogLevel info
  <IfModule mod_rewrite.c>
    RewriteEngine on
    # rewrite rules go here
  </IfModule>
  <IfModule php5_module>
    php_value magic_quotes_gpc off
  </IfModule>
  <IfModule mod_fcgid.c>
    AddHandler fcgid-script .fcgi
  </IfModule>
  AddHandler cgi-script .cgi
  Alias /cgi-bin/ /var/www/vhosts/www.somedomain.com/cgi-bin/
  <Directory /var/www/vhosts/www.somedomain.com/cgi-bin/>
    Options ExecCGI FollowSymLinks Includes
    AllowOverride Limit
    Order allow,deny
    Allow from all
  </Directory>
</VirtualHost>
```

Movable Type

- The following file should be *appended* to /var/www/vhosts/www.somedomain.com/cgi-bin/mt-config.cgi

```
AdminScript mt.fcgi
CommentScript mt-comments.fcgi
TrackbackScript mt-tb.fcgi
SearchScript mt-search.fcgi
ViewScript mt-view.fcgi
```

Appendix B: Change Log

Document Version	Description
1.2	Added section about using Rsync and NFS to keep systems in sync with one another.
1.1	Added section about the Movable Type Publish Queue. Stubbed out section for performing a Movable Type server migration.
1.01	Minor revisions.
1	Initial release. Primary section includes installation instructions for CentOS