Polopoly monitoring and tuning
- a system operator’s guide

Overview 2
Polopoly system metrics and monitoring 2
Checking and tuning cache parameters 6
  Checking the cache values 6
  Tuning the values 8
  Cache control 9
  Raw mode for monitoring 10
Finding and fixing choke points 11
  General approach 11
  The timer layers 12

Overview

The main challenge running distributed systems like Polopoly is to achieve a bird's eye view of the whole system. A disturbance in the database, or a depleted JDBC pool in JBoss might, in a worst case scenario, affect the performance of the whole installation. A problem that is reported as slow NewsLetter mailing might very well be traced to a slow LDAP connection from the UserServer module, from which the recipients are fetched.

The key is to monitor all systems from one place, so it is possible to visually get a feeling of how the components interact. A sudden drop in the number of served requests might be traced right away to a malfunctioning router, provided the graphs for requests and dropped Ethernet packages are in the same system view.

This document describes the most important aspects of the Polopoly system and how to read and tune it.

Polopoly system metrics and monitoring

It is important to setup monitoring of critical values in Polopoly to get a sense of how the system behaves and to learn the normal behavior over days and weeks. Only with historical system metrics is it possible to know if a reconfiguration or new deployment had any effect on performance.

There are several solutions available for gathering system metrics and presenting them in a digested and human readable format. Several of Polopoly's customers use the open source project Munin[1], which is a front end to RRD Tool[2] and this tool is also what the examples in this document will use.

The main system components that need monitoring are

- Network and I/O

  Network problems directly affect the performance of Polopoly.
  If connections hang, or are slowed down by malfunctioning interfaces it will have an influence on Polopoly. In fact, Polopoly handles broken connections ok, but a hanging, or slow, connection cannot (yet) be detected.

  Measuring the amount of traffic that passes different interfaces can also provide important clues if the system is inexplicably slow. If the front caches are too small, or constantly missed by a large amount of requests it will inevitably cause a raise in the traffic that passes the backend network interfaces that connect front and Content Management Server.

[1] http://munin.projects.linpro.no
[2] http://oss.oetiker.ch/rrdtool/

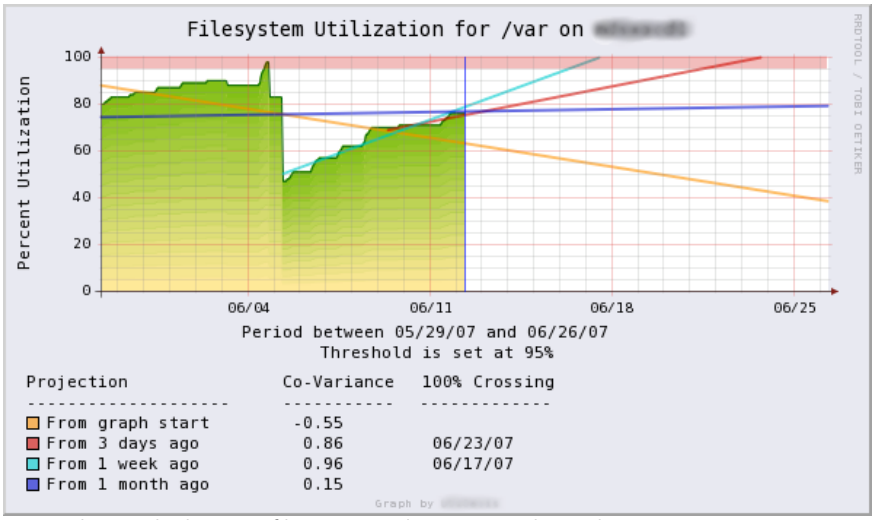
Relating to I/O is also the number of *concurrently open files*. Most operating systems treat files and sockets the same way (as files) and Polopoly can potentially need a lot of open files, as it accesses persistent caches and communicates with other modules. It is important to make sure that the number of open files is always well below the configured maximum for the process and it's equally important to make sure that the process has a high enough maximum for normal operations (with headroom for peaks). Note that the open files limit is configured on the operating system level and not in Java.

- CPU and memory

  Needless to say, it is important to monitor the physical resources on the machines used. CPU usage is vital, but also *memory swap activity*. JVM performance is dramatically reduced by swapping. It is crucial that the entire JVM can fit in RAM memory and that it is never (or very rarely) swapped out to disk.

- Disk space

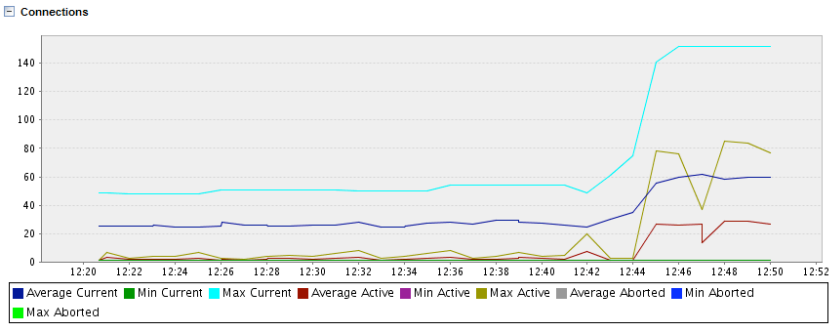
  Polopoly stores all content in a database, but also requires disk space for:
  - Persistent caches on the front servers
  - The Statistics module
  - The NewsLetter module
  - The Voting & Rating Module
  - The Lucene Search indices.
  - The database table space.



Example graph showing file space utilizations and trends3.

[3] Image stems from the show case gallery at the RRDTool site: http://oss.oetiker.ch/rrdtool/gallery/

- **Database statistics**
  As the main content storage, it is critical to monitor the health of the database[4]. Among the parameters these are most important:
  - Active threads
  - Queries/second
  - Slow queries
  - Available table space



Munin graph showing a sudden increase in database connection usage.

- **Data source connection pool usage (JBoss)**
  Of extra interest are active and idle connections. The details and the exact resource names used to get these values are container specific, but in JBoss4 the JMX name of the database pool is

  ```
  jboss.jca:name=<JDBC POOL NAME>,service=ManagedConnectionPool
  ```

  Use the JBoss JMX console `(http://localhost:8088/jmx-console/)` to inspect and explore the available MBeans. The port used here is dependent on how JBoss was installed. The attributes to monitor are
  - `InUseConnectionCount`
  - `MaxConnectionsInUseCount`
  - `ConnectionCount`
  - `ConnectionCreatedCount (not critical)`
  - `ConnectionDestroyedCount (not critical)`

- **Transaction manager statistics (JBoss)**
  The TransactionManager keeps track of the number of successful and roll-backed transactions, as well as the number of currently executing transactions in the EJB container. In JBoss, the JMX object name for the TransactionManager is

  ```
  jboss:service=TransactionManager
  ```

  Use the JBoss JMX console `(http://localhost:8088/jmx-console/)` to inspect and explore the available MBeans. The attributes to monitor are
  - `TransactionCount`
  - `CommitCount`
  - `RollbackCount`

[4] There are ready made Munin plug-ins for MySQL and Oracle downloadable from http://muninexchange.projects.linpro.no.

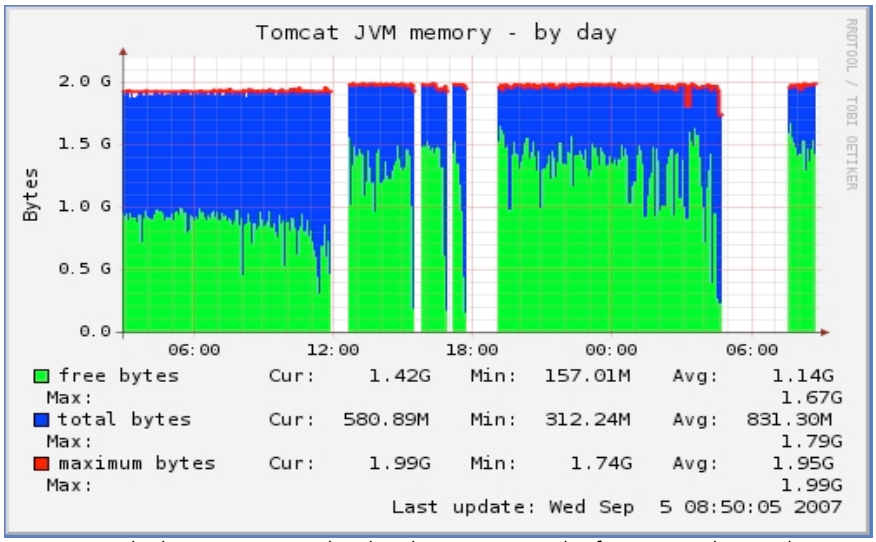
Especially watch out for TransactionCounts that are close to the maximum number of connections in the data source's connection pool, and for rapidly increasing RollbackCount.

- Entity bean cache statistics (JBoss)
  Each type of EJB entity bean has a separate bean instance cache. In JBoss this cache is a org.jboss.ejb.plugins.LRUEnterpriseContextCachePolicy, and it is configured through the jboss.xml deployment descriptor inside the ejb-jar. The most important entity types to monitor are ContentLocal and ContentMetaDataLocal, but for personalized sites, also UserDataLocal can be interesting. Their JMX object names are

```
jboss.j2ee:jndiName=ejb/ContentLocal,service=EJB
jboss.j2ee:jndiName=ejb/ContentMetaDataLocal,service=EJB
jboss.j2ee:jndiName=ejb/UserDataLocal,service=EJB
```

  Use the JBoss JMX console (http://localhost:8088/jmx-console/) to inspect and explore the available MBeans. The attributes to monitor are
  - CacheSize
  - CreateCount
  - RemoveCount

- JVM statistics
  JVM statistics are vital, especially memory for each running JVM. Look out for excessive garbage collections, as they might indicate that the process requires memory close to the maximum heap limit.



Munin graph showing a JVM that has been restarted a few times during the period.

- Polopoly front caches
  The details are described further down in this document, but of extra importance are
  - Content cache hit ratio (cache efficiency).
  - Policy cache hit ratio (cache efficiency)
  - Size of the content cache.
  - Size of the Policy cache.

Checking and tuning cache parameters

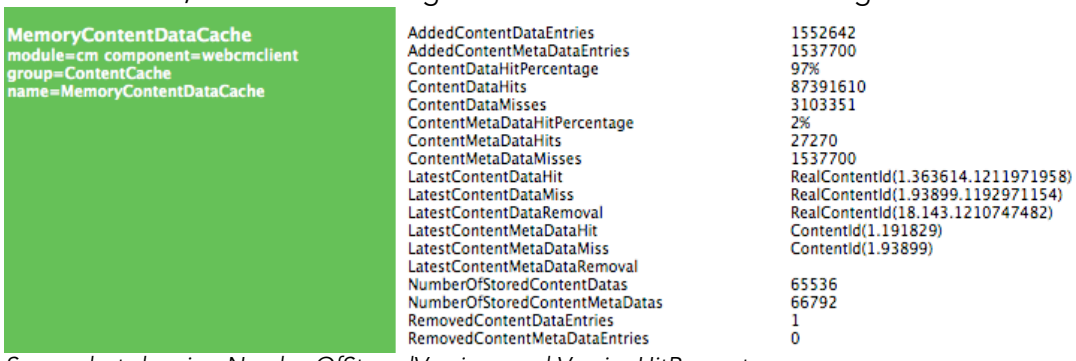
Polopoly relies on the JMX[5] protocol for monitoring and management. Most of the Polopoly core functions can be accessed and read using this protocol[6]. Please refer to the section "JMX Management and Monitoring" in the Polopoly Developer's Guide for an introduction to JMX and Polopoly.

Checking the cache values

Of special interest, from a caching point of view, is the Polopoly Status page that exposes the Polopoly caches over HTTP. The page is valuable from several aspects, but can appear slightly daunting as it exposes *all* values that can be read from the Polopoly caches. For normal operations only a few of those values are worth monitoring. Selecting "`<your hostname> - front - FINE`" in the initial select box (top, right on the status page) will present the most interesting page. The values are updated in real-time using AJAX.

Checking the content cache

The content cache holds the actual in-memory content data (excluding files). There most interesting values are *NumberOfStoredContentDatas and ContentDataHitPercentage* respectively. Those indicate how much content is loaded in memory and the resulting cache efficiency[7] measured in percent. Typically, the cache size should be tuned to get as high hit percentage as possible, while keeping within the bounds of JVM memory (tune and check the JVM graph before and after). Suitable values vary widely depending on what Polopoly is used for and the traffic patterns on the site/sites installed, but it is not uncommon to see values in the range of 90.000 – 150.000 memory cached content datas, which is much larger than the current default configuration[8].



Screenshot showing NumberOfStoredVersions and VersionHitPercentage.

The data in the memory cache stems from content or user data (personalization information). Both caches need monitoring and the attributes to monitor are `NumberOfStoredContentDatas` and `ContentDataHitPercentage`. The full JMX names are:

---

[5] Java Management eXtensions. See http://java.sun.com/developer/technicalArticles/J2SE/jmx.html for an overview.

[6] There is a JMX-plugin for Munin, which is used to fetch the values from Polopoly. See http://muninexchange.projects.linpro.no

[7] It is actually slightly more complicated than described, as Polopoly keeps several in-memory caches for different aspects of the system, but the values presented here are central.

[8] Version 9.7 and version 9.9

- `com.polopoly:host=<YOUR HOST HERE>,application=front,module=cm, component=webcmclient,detailLevel=FINE,group=ContentCache, name=MemoryContentDataCache:NumberOfStoredContentDatas`

- `com.polopoly:host=<YOUR HOST HERE>,application=front,module=cm, component=webcmclient,detailLevel=FINE,group=ContentCache, name=MemoryContentDataCache:ContentDataHitPercentage`

Checking the policy cache

The policy cache holds the java content wrappers (Policy-instances) which refers the content data. These caches form quite big java instance trees and although it is efficient to hold these structures in memory, it can be a resource hog that eats too much memory. A balance has to be struck, trading CPU (instance creation) for memory (speed) up to a certain point. In this case, bigger is not always better and there have been cases where memory problems have been solved by shrinking the size of the policy cache, without affecting performance in any significant way.

The attributes to check and monitor are:
`PolicyCacheHits, PolicyCacheMisses, PolicyCacheBypasses` and `PolicyCacheSize`.

Note that cache hit ratio is not computed, but may be calculated using the formula:
`Hit Ratio = 100 * PolicyCacheHits / (PolicyCacheHits + PolicyCacheMisses + PolicyCacheBypasses)`

The MBean name is:

```
com.polopoly:host=<YOUR HOST HERE>,application=front,module=cm,
component=<COMPONENT NAME>,detailLevel=INFO,group=PolicyCMServer,
name=PolicyCache
```

Checking the external-id cache

The external ID cache is important when the application refers content by their symbolic external ID strings. External ID strings are often used when content is XML imported or exported.. For other majors, custom Polopoly applications use external IDs in various degrees. Call statistics from the "Cache Timer" layer can be used to find out how much external IDs are used in your particular Polopoly application.

The cache maps each ID string a regular content ID (major/minor).

MBean object names:
`com.polopoly:<YOUR HOST HERE>,application=front,module=cm, component=webcmclient,detailLevel=FINE,group=ContentCache, name=MemoryExternalIdLookupCache`

Important MBean attributes to monitor are `HitPercenatage` and `Size`

Checking the lookup cache

The lookup cache is used very frequently, and thus performance is critical. It maps symbolic ContentIds to RealContentIds. For example, it can map "currently public version of 1.2134" to "1.2134.18728746589".

MBean's object name:

```
com.polopoly:<YOUR HOST HERE>,application=front,module=cm,
component=webcmclient,detailLevel=FINE,group=ContentCache,
name=MemoryContentLookupCache
```

Important MBean attributes to monitor are: `HitPercentage` and `Size`

Tuning the values

The cache values can be changed by configuration. This section summarizes the most important parameters to configure.

Tuning the content cache

The size of the content memory cache is controlled by a single parameter. To change it, add the following lines to your configuration:

```
default.webcmclient.ContentCache.contentMemoryCacheSize=100000
default.polopolycmclient.ContentCache.
contentMemoryCacheSize=100000
```

For the persistent caches, sizes are unlimited, but the placement is configurable. It is worth noting that the default location of the persistent caches is standards compliant, but might be sub optimal in other regards. The default location is in the web applications' directory, which means that the cache might be cleared every time the web application is redeployed. This forces the front server to get it all from the backend server after redeployment, which in turn puts unnecessary load on the Content Management server.

Polopoly recommends changing the configuration of the persistent cache and setting a directory outside the web container. This is done using the configurations listed below. Should you wish to turn off the persistent cache, set the values to the empty string. This will cause Polopoly not to use persistence caches at all.

When setting up the persistent cache, it is also important to make sure that:

1. The cache resides on local disk (and not NFS).
2. There is enough room to govern the cache. 10 – 100GB should be fine.
3. Extra performance can be achieved if `atime` is deactivated in the OS.
4. A unique directory must be used per host and application.

Cache directories can be configured using:

```
default.webcmclient.DiskCache.contentCacheDirectory=<dir0>
default.webcmclient.DiskCache.filesCacheDirectory=<dir1>
default.polopolycmclient.DiskCache.contentCacheDirectory=<dir2>
default.polopolycmclient.DiskCache.filesCacheDirectory=<dir3>
```

Tuning the policy cache

The maximum size of the policy cache can be configured by a single parameter for each PolicyCMServer. The two most important config instances of PolicyCMServer in the standard ConfigInstances.properties are `webcmclient` (used in the front server webapp) and `polopolycmclient` (used in the editorial GUI webapp). Configuration examples:

```
default.webcmclient.ContentCache.policyMemoryCacheSize=10000
default.polopolycmclient.ContentCache.policyMemoryCacheSize=10000
```

Tuning the external id cache

The maximum size of the external ID cache can be configured with a property called externalIdMemoryCacheSize. The `polopolycmclient` is used in the editorial GUI, while the `webcmclient` is used in the front web application.

```
default.webcmclient.ContentCache.externalIdMemoryCacheSize=80000
default.polopolycmclient.ContentCache.externalIdMemoryCacheSize=80000
```

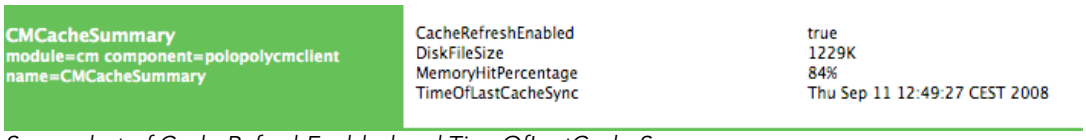
Tuning the lookup cache

The maximum size of the lookup cache can be configured with a property called `lookupMemoryCacheSize`. Separate settings are used for the UserData major ("UserCache") and all other majors ("ContentCache"). The `polopolycmclient` is used in the editorial GUI, while the `webcmclient` is used in the front web application.

```
default.webcmclient.ContentCache.lookupMemoryCacheSize=70000
default.polopolycmclient.ContentCache.lookupMemoryCacheSize=70000
```

Cache control

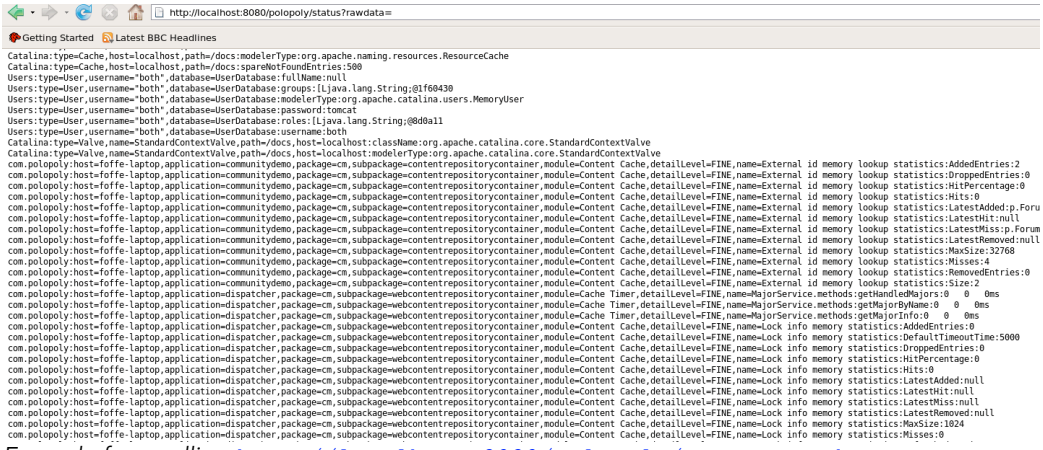
Caches are refreshed at a regular interval. The cache refresh can be automatically disabled if the front switches to autonomous mode. If the site is not updating with new content it is worth checking the *CMCacheSummary* of the Polopoly Status page. Values to check are *CacheRefreshEnabled* and *TimeOfLastCacheSync.* The first of those two should show *true* (update enabled) and the second should update regularly as the front updates itself. If they don't, some kind of error has occurred and a deeper analysis is required, which should start with going over all system graphs to check that they are within normal limits.



Screenshot of CacheRefreshEnabled and TimeOfLastCacheSync.

Raw mode for monitoring

The status page can also be used for monitoring these values using HTTP. It is done passing a "rawdata" parameter to the Polopoly Status Page[9]. Passing an empty argument will yield a complete dump of all JMX attributes. Example: http://localhost:8080/polopoly/status?rawdata=



Example from calling http://localhost:8080/polopoly/status?rawdata=

Using this method, it is possible to filter out any JMX exposed value/values from the JVM, including the Polopoly cache values. By passing arguments with the rawdata parameter, it is possible to filter and pick out specific values directly. The argument is the JMX-name of the parameter, which are also the same names that are listed in the complete dump. It is just a matter of finding the value you want and copy-pasting it in the browser navigation to test.
Example for getting the `AddedContentDataEntries` value using rawdata (on localhost):

```
[...]/status?rawdata=com.polopoly:host=localhost,application=front,module=
cm,component=client,detailLevel=FINE,group=ContentCache,name=MemoryContent
DataCache:AddedContentDataEntries
```

It is also possible to use the wildcard '*' to filter out a block of values. As in:

```
[…]localhost:8080/polopoly/status?rawdata=com.polopoly:host=my-laptop,*
or
http://localhost:8080/status?rawdata=com.polopoly:detailLevel=SUMMARY,*
```

[9] This functionality was added in version 9.7.4 of Polopoly.

Finding and fixing choke points

Writing the complete guide to problem solving is difficult and most probably impossible. There are however a few general guidelines which can be followed and some tricks as well. This part of the document aims to provide you with a basic understanding of the approach to sorting out problems with a malfunctioning Polopoly installation.

General approach

The general approach to finding and fixing choke point is to:

1. *Know your normal system behavior (a prerequisite!).*
   If you do not know the normal, you will not spot the abnormal; therefore it is very important to setup proper monitoring and keeping system statistics of the installation using tools like Munin.

2. *Check resources for starvation.*
   Lags, sluggishness and mysterious halts can usually be attributed to resource starvation. Any resource that hits its maximum value has the potential to cause a ripple effect and slow down or halt the installation. Disk space, CPU, the number of concurrently open files, I/O-wait, … they are all critical for performance.

   Tools like `top`, `netstat and lsof` are invaluable for inspecting the runtime performance of a sluggish sever.

3. *Thread dumps and logging.*
   If all system resources seem ok (on *all servers involved*), chances are that some part of the system is malfunctioning. It may be down to environment, configuration or a bug. To find the cause you need to hypothesize and then systematically verify, or falsify the hypothesis.

   You do this mainly by going through the logs for errors and by thread dumping the process under inspection. Polopoly uses standard Java logging[10] and thread dumps are preferably done using `kill -QUIT <pid>`. Dumps are most valuable when a few of them are taken in sequence, with a few seconds delay in between.

   From the log inspection and thread dumps, a number of suspect classes will most likely emerge. For those, logging should be adjusted to increase verbosity from the relevant packages. This can be done runtime using JConsole, which is documented at `http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html`.

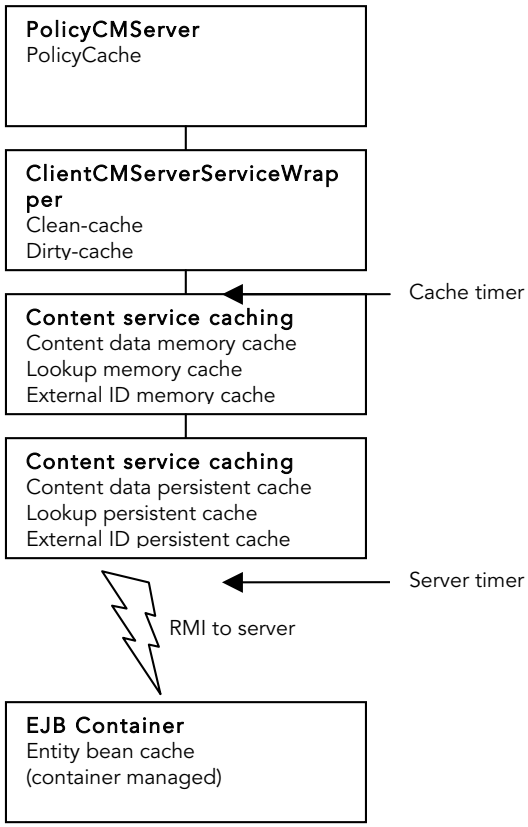
[10] See http://java.sun.com/j2se/1.5.0/docs/api/java/util/logging/LogManager.html

Polopoly has an improved log-formatter, called PolopolyFormatter[11] that is recommended. Since it must be loaded by the Java's system class loader, make sure to drop `polopolyformatter.jar` in `$JAVA_HOME/jre/lib/ext/`.

Thread dumps contain the entire JVM state at the time of the dump. Analyzing them can be cumbersome and requires some skills to get it right. What you look for is irregular patterns like a lot of threads that are waiting for the file system, or threads that are locked and waiting for each other (possible deadlock).

The timer layers

Pages or web applications may be slow due to suboptimal use of the Polopoly APIs. The Polopoly connector that talks to the Content Management Server from the client side[12] has a stacked architecture, where each layer in the stack adds a specific service to the connector. The services consist of different caches and a wrapping service for creating Policies automatically from content.



These layers are often referred to as the content repository stack. In two places of the content repository stack, there are *timer layers*. These layers have two tasks: counting the number of times each service method has been called, and keeping track of how long each request takes. One timer layer sits on top of the content cache ("Cache Timer") and one sits below the cache ("Server Timer"). The two timing layers are aware of each other, and by comparing the request count, hit ratio can be calculated (shown in the Cache Timer's data).

Looking at these counts and timing values provide valuable information about how the Polopoly API is used and how much time is spent in the connector layer. The timer layers are by default deactivated to gain performance, but when detailed information is desired, they can be activated, using configuration, or runtime using JMX and MBeans.

[11] The full name is com.polopoly.util.logging.impl.PolopolyFormatter.

[12] The clients are typically the web applications (deployed in Tomcat), or a separate modules, like the Index Server.