Gathering CVMFS statistics

September 2018

Author:
Daniel-Florin
Dosaru

EP-SFT
CernVM-FS

Supervisors:

Radu Popescu
Jakob Blomer

Project Specification

The CernVM File System (CernVM-FS) provides a global software distribution service implemented as a POSIX read-only filesystem in user space (FUSE). It is a mission critical component for the LHC experiments and many other HEP collaborations. In CernVM-FS, files are stored remotely as content-addressed blocks on standard web servers and are retrieved and cached on-demand. For writing, CernVM-FS follows a publish-subscribe pattern with a single source of new content that is propagated to the world-wide network of reading clients.

This project involved implementing, in the CernVM FS server tools, a system for gathering useful statistics about the publication process. The information gathered gives insight into the contents of the published payloads (number of files, payload size, type of files published), as well as some meta-information (garbage collection statistics, duplication information if available). The statistics are stored locally, on the release manager machine, in a SQLite file data format to facilitate analysis. The gathering system can also analyze these local data stores individually or by aggregating the data stores from different release manager machines (cvmfs_server merge-stats).

The statistics can also be submitted to Graphite, a free open-source software tool that monitors and graphs numeric time-series data.

Contents

Acknowledgements

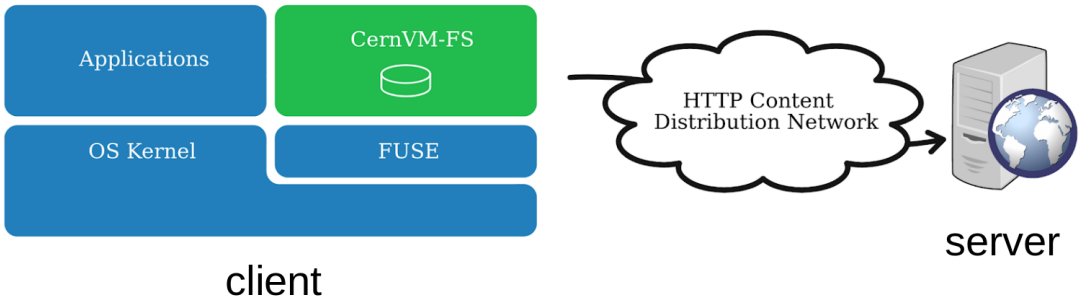
I would like to thank both of my supervisors, **Radu Popescu** and **Jakob Blomer**, for their great support during my entire stay at CERN and for always being there to answer any questions that came up in the development process. This experience was very productive for me and I learnt a lot of new things from them. I would also like to thank them for selecting me in the first place and putting the trust in me to work on this project.

In general, I would like to thank the entire EP-SFT department as well as the CERN Summer Student Team for allowing me to spend my summer at CERN in Geneva working on this exciting project and for their support during the stay.

1. Introduction

The CernVM-File System (CernVM-FS) provides a scalable, reliable and low-maintenance software distribution service. It was developed to assist High Energy Physics (HEP) collaborations to deploy software on the worldwide-distributed computing infrastructure used to run data processing applications.

CernVM-FS is implemented as a POSIX read-only file system in user space (a FUSE module). Files and directories are hosted on standard web servers and mounted in the universal namespace /cvmfs. Internally, CernVM-FS uses content-addressable storage and Merkle trees in order to maintain file data and meta-data. CernVM-FS uses outgoing HTTP connections only, thereby it avoids most of the firewall issues of other network file systems. It transfers data and meta-data on demand and verifies data integrity by cryptographic hashes.



By means of aggressive caching and reduction of latency, CernVM-FS focuses specifically on the software use case. Software usually comprises many small files that are frequently opened and read as a whole. Furthermore, the software use case includes frequent look-ups for files in multiple directories when search paths are examined.

CernVM-FS is actively used by small and large HEP collaborations. In many cases, it replaces package managers and shared software areas on cluster file systems as means to distribute the software used to process experiment data.

In contrast to general purpose network file systems such as NFS or AFS, CernVM-FS is particularly crafted for fast and scalable software distribution. Running and compiling software is a use case general purpose distributed file systems are not optimized for. In contrast to virtual machine images or Docker images, software installed in CernVM-FS does not need to be further packaged. Instead it is distributed and versioned file-by-file.

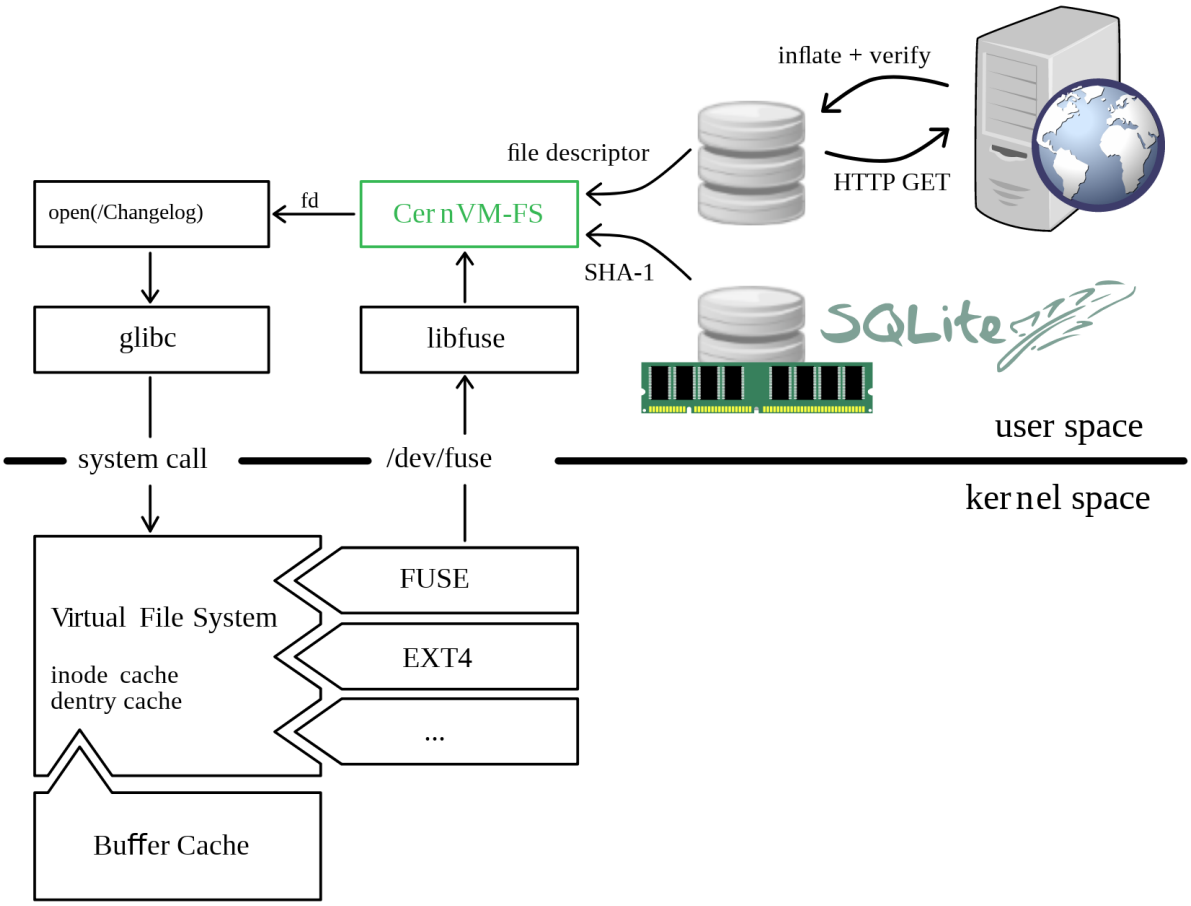


Figure 1.1: *On the client, only data and metadata of the software releases that are actually used are downloaded and cached.*

2. Repository publishing

In order to create and update a repository, a distinguished machine, the so-called ***Release Manager Machine***, is used. On such a release manager machine, a CernVM-FS repository is mounted in read/write mode by means of a union file system. The union file system overlays the CernVM-FS read-only mount point by a writable scratch area. The CernVM-FS server tool kit merges changes written to the scratch area into the CernVM-FS repository. Merging and publishing changes can be triggered at user-defined points in time; it is an atomic operation. As we can see, a CernVM-FS repository is similar to a versioning system repository.

Repositories are periodically updated: installation of a new release or a patch for an existing release. Each time only a small portion of the repository is modified. In order not to re-process an entire repository on every update, CVMFS has a read-write file system interface to a CernVM-FS repository where all changes are written into a distinct scratch area. Union file systems combine several directories into one virtual file system that provides the view of merging these directories. These underlying directories are often called branches. Branches are ordered; in the case of operations on paths that exist in multiple branches, the branch selection is well-defined. By stacking a read-write branch on top of a read-only branch, union file systems provides a writable interface for the file system.

Union file systems can be used to track changes on CernVM-FS repositories (Figure below). In this case, the read-only file system interface of CernVM-FS is used in conjunction with a writable scratch area for changes.

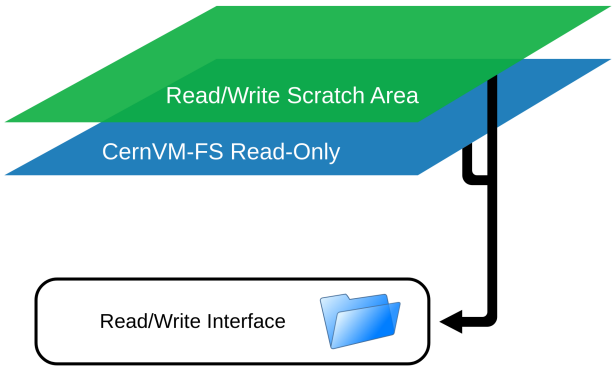


Figure 2.1: *A union file system combines a CernVM-FS read-only mount point and a writable scratch area. It provides the illusion of a writable CernVM-FS mount point, tracking changes on the scratch area.*

Change sets are published to CernVM-FS repositories through a transactional interface. Committing a transaction will increase the repository's revision number by 1 and make the new changes available to all repository clients.

3. Contributions

3.1 Publish pipeline

The publish pipeline (see figure below) is started by one of the following cvmfs_server commands: either ingest, which extracts the content of a tarball into a repository, either publish, which creates a new repository snapshot by applying the changes made after the start of the transaction.

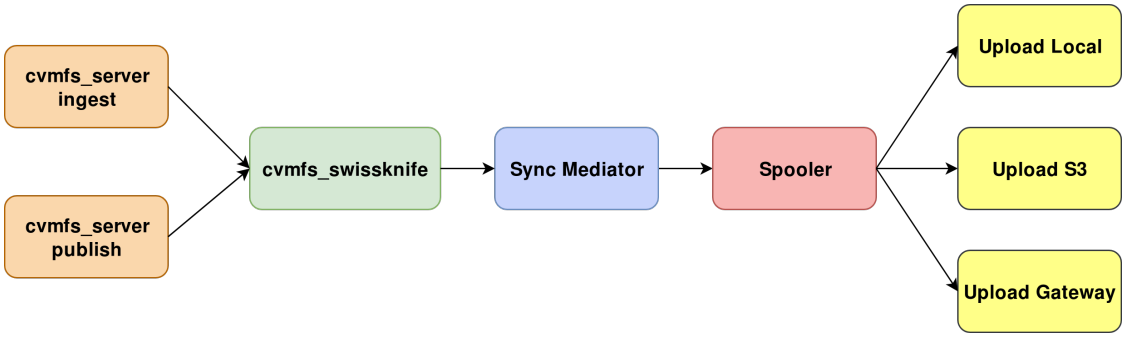


Figure 3.1: Publish pipeline

The cvmfs_swissknife binary identifies the command received and triggers the proper action for each command. In our case, the sync (publish) and ingest commands are shown. Both of these call the sync mediator element of the pipeline.

The SyncMediator is an intermediate layer between the UnionSync implementation and the CatalogManager. Its main responsibility is to traverse newly created directories and to add all included files. Also, new and modified files are piped to the Spooler.

The SyncMediator is a good point to count the following metrics:

- number of added/removed/changed files
- number of added/removed/changed directories
- number of added/removed bytes

The Spooler is the entry point to the general file processing facility of the CVMFS backend. It works with a two-stage approach:

1. Process file content
    - create smaller file chunks for big input files
    - compress the file content (optionally chunked)
    - generate a content hash of the compression result
2. Upload the file (it supports different upload paths: local, S3, ...)

There are a number of different entities involved in the processing of a file content:

- Spooler - general steering tasks (+ common interface)
- IngestionPipeline - chunking, compression and hashing of files
- AbstractUploader - abstract base class for uploading facilities
- Concrete Uploaders - upload functionality for various backend storages

At the AbstractUploader level and at the Concrete Uploaders I added my contribution for counting the compressed data size and the number of duplicated objects found.

```
[root@phsftpc2r20:~]# cvmfs_server transaction test.cern.ch
[root@phsftpc2r20:~]# # add changes to /cvmfs/test.cern.ch
[root@phsftpc2r20:~]# cvmfs_server publish test.cern.ch
```

Figure 3.2: Publish workflow in a simple scenario

3.2 Garbage collector

The GarbageCollector is figuring out which data objects (represented by their content hashes) can be deleted as outdated garbage. Garbage collection is performed on the granularity of catalog revisions, thus a complete repository revision is either considered to be outdated or active. This way, a mountable repository revision stays completely usable (no nested catalogs or data objects become unavailable). A revision is defined by its root catalog; the revision numbers of nested catalogs are irrelevant, since they might be referenced by newer (preserved) repository revisions. This way, garbage objects are those that are **not** referenced by any of the preserved root catalogs or their direct subordinate nested catalog try.

It use a two-stage approach:

1. **Initialization**: The GarbageCollector is reading all the catalogs that are meant to be preserved. It builds up a filter containing all content hashes that will **not** be deleted
2. **Sweeping**: The initialized filter is presented with all content hashes found in condemned catalogs and decides if they are referenced by the preserved catalog revisions or not.

Figure 3.3: Garbage collector pipeline

The Garbage collector pipeline follows a similar pattern (similar to the publish pipeline in the first two blocks): cvmfs_server gc command uses cvmfs_swissknife binary and then it calls the collect function from the GarbageCollector class which:

- **Analyze the preserved CatalogTree**
- **Checks the Preserved Revisions**
- **Calls the Sweep function**

In order to count number of preserved catalogs, number of condemned catalogs, number condemned objects and optionally the number of condemned bytes I changed the **Sweeping phase** by increment the thread safe counters.

I made optional the count of the condemned bytes because this might increase the garbage collector runtime. This measurement uses the stat system call many times which means many context switches that leads to some overhead, but further test should be run in order to evaluate the actual impact.

In order to activate this optional measurement **CVMFS_EXTENDED_GC_STATS** variable needs to be set in the server.conf file.

```
[root@phsftpc2r20:~]# cvmfs_server transaction test.cern.ch
[root@phsftpc2r20:~]# # remove files from /cvmfs/test.cern.ch
[root@phsftpc2r20:~]# cvmfs_server publish test.cern.ch

[root@phsftpc2r20:~]# cvmfs_server gc test.cern.ch
```

Figure 3.4: Garbage collector usage in a simple scenario

All the statistics are stored locally in a SQLite file.

In order to change the location of the SQLite file **CVMFS_STATISTICS_DB** variable needs to be changed. The default location is "/var/spool/cvmfs/REPO_NAME/stats.db" .

4. Use Cases

4.1 Monitoring CVMFS

This project can be used as a monitoring tool for cvmfs. send_stats_to_graphite.py script should be called to send only the new stats data from the local database file to a tool that monitors and graphs numeric time-series data such as Grahite.

4.2 Alarm system

Different conditions can be verified in order to create an alarm system to discover unusual behaviour of the CVMFS. For example, a certain action could be started if no garbage collector process was registered. As an early success story for my project I can mention that on the 21st of August the garbage collector did not run (see figure 5.1), so I investigate the problem and I found out that the scripts that calls the garbage collector were updated that day.

4.3 Benchmarking tool

By recording the start time and finished time of each publish or garbage collector action my project can help developers and the users of CVMFS to improve its performance. As we can see in the LHCb nightly test release manager statistics (see figure 5.5), the garbage collector run time is about two hours in average and it might be a good idea to reduce this in the future.

5. LHCb nightly test release manager statistics

The system for gathering statistics was installed on LHCb nightly test release manager and it recorded data for one week in the following time interval: 2018-08-15 00:00 - 2018-08-21 23:59 (CEST).

5.1 Publish activity over the week

In figure below it is shown publish activity over a week, every day more than three millions of files are added or removed.

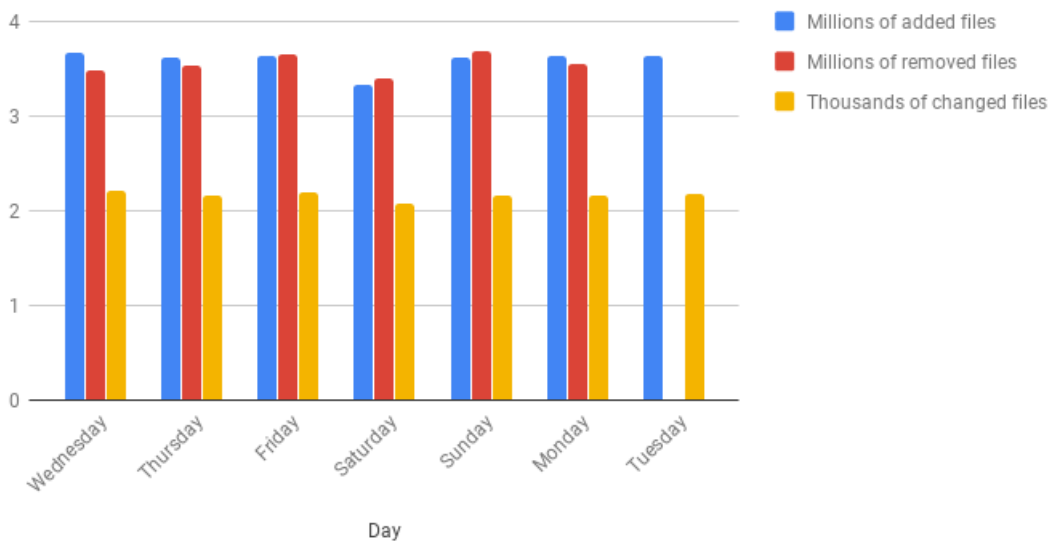


Figure 5.1: As it was mentioned before, Tuesday, on the 21st of August, no file was removed due to the update of the scripts that do this job

5.2 Publish activity over the day

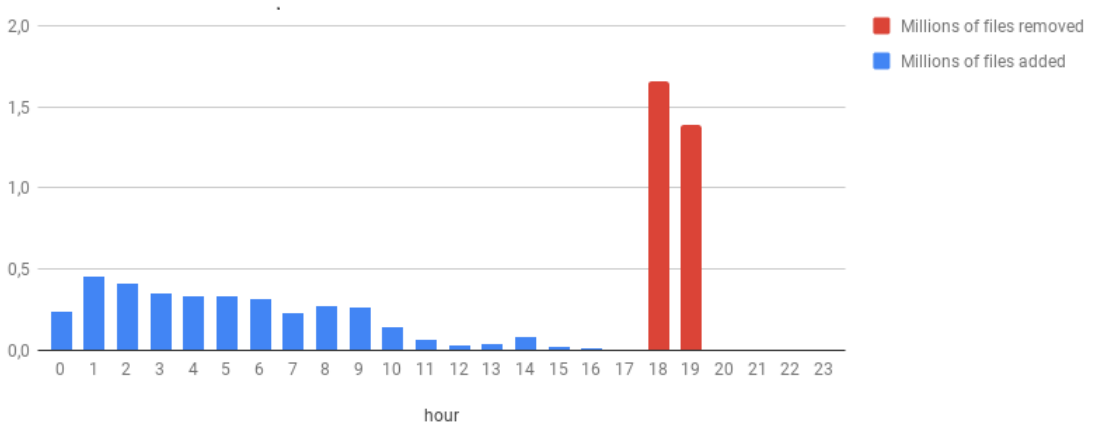


Figure 5.2: In the first part of the day new files are added to the repository and after 6pm some files are removed in a normal day.

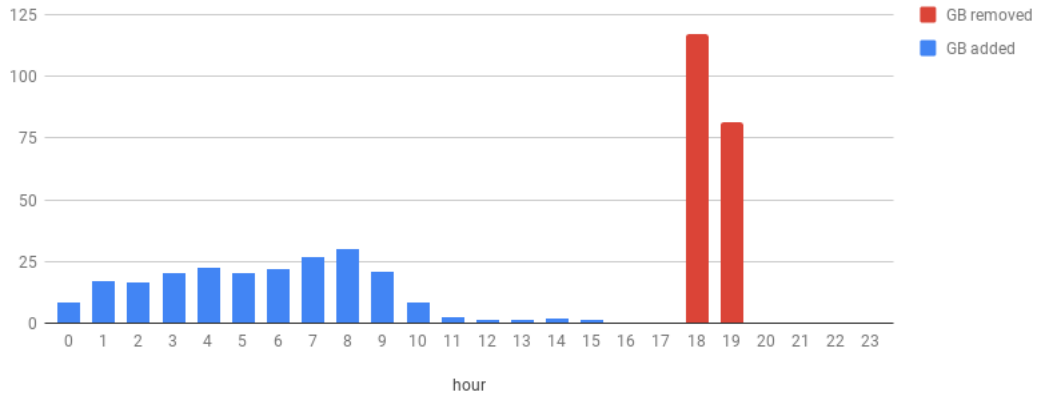


Figure 5.3: Data size added/removed

5.3 Garbage collector activity over the week

The average run time for the garbage collector process is more than two hours and if it does not delete files daily it can reach even three or four hours. (See figures 5.4 and 5.5)

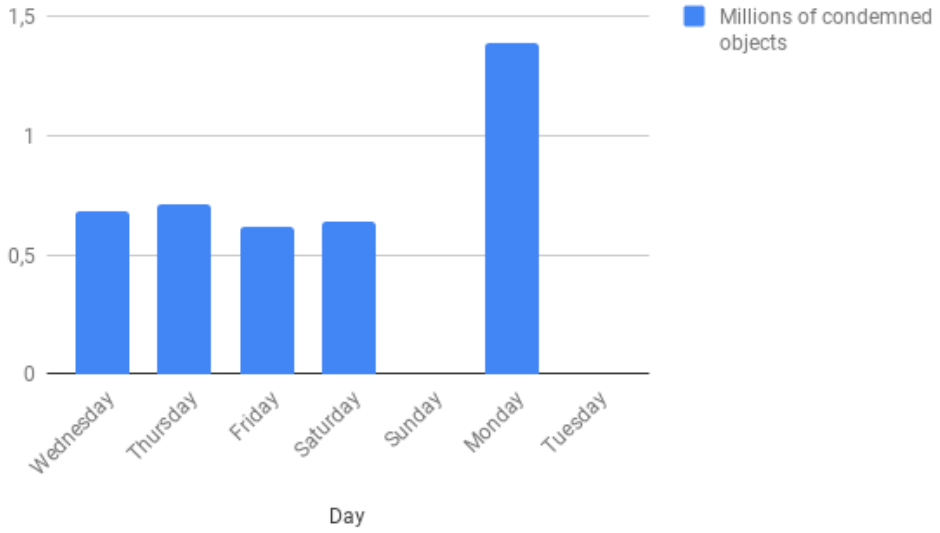


Figure 5.4: Garbage collector activity: condemned objects

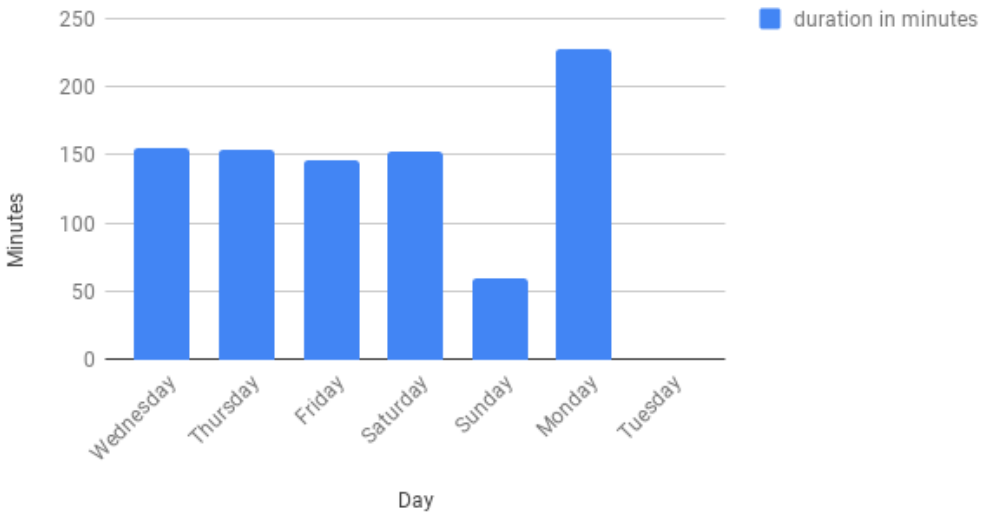


Figure 5.5: Garbage collector run time

5.4 Other interesting numbers

Average number of transactions per day	1,070
Average number of added files per transaction	3,286
Maximum number of files in one transaction	51,306 (746 MB)
Maximum data size added in one transaction	2 GB (for 2,133 files added)
Maximum run time for a publish command	4 minutes

6. Conclusions

6.1 A monitoring tool

Gathering CVMFS statistics project created a monitoring tool for CernVM FS server tools by recording different metrics into a local SQLite file:

- number of added/removed/changed files
- number of added/removed/changed directories
- number of added/removed bytes
- compressed data size added/removed
- number of duplicated objects added

It will be available in the CernVM-FS 2.6.0 release.

6.2 Future work

- More metrics can be added to this project in order to make use of the already implemented gathering statistics system used to record current metrics.
- A Grafana instance can be used to visualize the data in a more user friendly environment.
- Further test should be made to measure the cost of counting the number of condemned bytes in the garbage collector process.

A. Source code and documentation

The implementation for this project can be found on **https://tinyurl.com/gathering-cvmfs-stats**.

More information about the CernVM File System can be found on this documentation: **https://cvmfs.readthedocs.io**.