Sympa Performance Issues

The resource of a software come from these three components:

1. Memory usage
2. CPU usage
3. Disk usage

However there is coherence between this resources I try to handle these as separate parts of the project. But if a modification has a big impact in one component but this decrease the overall performance I have to doubt it.

Memory usage

General

- Perl memory management has a weird behaviour: it doesn't free memory even the process exited from the scope (subroutine etc.) . If big objects are created than they have to be *undef*ed at the and of scope.

<box green|Remarks by David> It's not supposed to be that way. The documentation on perl object is clear on the fact that perl is supposed to handle the object destruction and, I suppose, subsequent memory managemnent, as it is the point of object destruction:

> “When the last reference to an object goes away, the object is automatically destroyed.. (This may even be after you exit, if you've stored references in global variables.)”

</box>

<box blue|Remarks by Andras> We have discussed about it. The reality is so far away that. “[..]Because a lot of the (existing) code does not use a explicit undef on a scope exit (of course, does this imply a variable declared with 'my').” source </box>

- I take same profiling with tool named Gladiator. It go through the writes out the number of SVs, hashes. I have put this probe in the beginning of the sympa.pl's infinit loop. After processing messages the number of SVs slightly increased. The reasons:
  - Global variables

<box green|Remarks by David> We want to remove global varialbes form the main daemons. That's a big work as we used them a lot, especially in wwsympa.fcgi, which has the worst memory usage. </box>

<box orange|Remarks by the team> Olivier did another perl web application that uses a “Request” object, whose job is especially to store incoming and outgoing data and to generate the request answer. You could use such an object to get rid of globals in wwsympa.fcgi. Here is an example of this code. </box>

- Static classes

<box green|Remarks by David> We are hunting them. One is down already: The DataSource and its sibblings, SQLsource and LDAPsource. Would you say that it is better to always use implemented object instead of static classes? If I'm not mistaken, the class structure is loaded in memory for further implementations, so using objects should increase memory consumption, no? </box>

<box blue|Remarks by Andras> No, I think static classes are better than always declare objects. But we have count on it when hunting for memory leaks. </box>

- Memory leaks

With the first two I can't do anything. Maybe regularly should be freed in the loop if didn't consists valid information. The third one is a serious problem I have to go deeper and check around the subrutines…

<box green|Remarks by David> That's great because we really nedd your help on the third one!</box>

bulk.pl

- The bulk.pl is one of the daemons which is leaking in our production environment. I haven't found the source of these leaks. I studies the spamassassin daemon which has a similar structure (multiprocess, all process the same task). The difference is that the spamd master daemon doesn't handle any messages, it only checks the conditions and forks if new daemon is needed. The master bulk daemon should do the same.

<box green|Remarks by David> Interesting, but I don't see for now how the messages sending can influence the ability of a daemon to fork into children. Is it because it holds data in memory (messages for example) and is connected to the dataabse? The fork is done at the start of the loop, just after having checked the number of messages waiting in the database spool, but before loading a message to process. Maybe we don't cleanly delete the message processed during the previous loop? </box>

- Bulk::messageasstring seems to be dead code.

<box green|Remarks by David> Indeed. Don't mind it, we have rewritten most of the message sending process (the changes are included in you SVN.) so dead code will be removed soon. </box>

<box orange|Remarks by the team> Most of the team (contrary to what I told you via Skype) agrees on the fact that moving the packet preparation to bulk.pl would not be a good idea. The process that prepares the sending packets (currently, sympa;pl) needs to access the lists config to have the VERP rate, subscribers options, and a lot of sending options. It does some pre-sending treatments (data merge, S/MIME and DKIM signing) because because they can't be done otherwhere, but all the packets preparatnios should still remain in Sympa. In addition, we are working on the Sympa clusterization, which implies making all ressources available and sharable by several processes. In such a configuration, we could have several sympa.pl running and preparing packets.

We think that the best rate effort / result for you would be obtained by focusing an the bulk.pl daemon modifications, the wwsympa.fcgi process (especially the global variables) and maybe the memory leaks in sympa.pl, as you could use the results of the experiments led by Olivier and reproduces at the bottom of this page. </box>

sympa.pl

bounced.pl

task_manager.pl

wwsympa.fgci/SOAP

- Terrible memory usage and leaking…supply

<box green|Remarks by David> Yes… Please, help us!😕 </box>

- 27th of April: After debugging a whole night, I found a memory leak in external Template module. Only the version 2.20 is affected, unfortunately Ubuntu 10.04 ships this version.:(

other

CPU usage

General

- Drop message priorities: prioritizing the message sounds good at the first time, but I don't see any advantages: a non-overloaded system every message get through the sympa in few seconds even at a large traffic server. On the other had classifying a message takes some resources and make the code more complex.

<box green|Remarks by David> Actually, messages priority is crucial in a lot of cases. For example, if you are distributing a newsletter to 10,000 subscribers, it can take several minutes. Sympa can interrupt the distribution to send a message to a higher priority list. This is important when it is a list ofr important people, such a the list of university presidents of the country, or an alert mailing list. Sometimes, you d'ont want, or just can't wait for distribution. Withou priorities, messages would be distributed roughly according to their arrival date, which is not related to the degree of urgency of the information distribution. In addition, when your Sympa is stopped for half a day, messages keep on being stored in the spools. It is a good thing that, once Sympa is restarted, The most important lists are handled first. </box>

bulk.pl

sympa.pl

- Cleanspool functions should be moved to task_manager.pl. They can hang the message process.

<box green|Remarks by David> Indeed. We plan to let the task_manager.pl daemon handle this. Cleanspool functions are a pre-task_manager era remnants and evolution should put them out. </box>

- The queue checking algorithm is O(n*n*log(n)) because every time the daemon process a message look into the spool for new messages. It should process all founded message in the spool.

<box green|Remarks by David> Are you sure? I remembered that we make a readdir operation in the spool, put all the messages found in a hash and then process these messages. Maybe I don't understand correctly.</box>

bounced.pl

task_manager.pl

wwsympa.fgci/SOAP

other

Disk usage

bulk.pl

sympa.pl

- The currently used filesystem queue model has two major disadvantages:

1. The “queue” C program is called for every recipient and it has to be written to the disk in n copy (where n is the number of recepients) and has to process every copy of the letter
2. sympa.pl doesn't know about the other recipients. It is hard to implement eg. a crosspost-filter

My suggestion: At the MTA→sympa communication implement LMTP. LMTP is an extension of ESMTP where the receiver can reject a letter on per recipient basis. So no need to write the message to the disk.

```
Disadvantages:
  * Limited queue: because of the sympa.pl is a single thread app, the queue is limited by the OS TCP queue
  * The priority system can't be implemented: the sympa.pl become a FCFS system.
```

<box green|Remarks by David> Do you mean that the send scenario evaluation should be done at

the queue level? that sympa.pl should replace the C queue program? If this is the case, I see a problem: For now, when sympa.pl is stopped, the messages still go to the spool and can be handled after. So implementing such a feature would expose us to the risk of losing messages arriving while sympa.pl is down. </box>

bounced.pl

task_manager.pl

wwsympa.fcgi/SOAP

other

<box orange|Remarks by the team> Olivier did a deep analysis of concerning brutal memory concumption rises. You can find all his analysis in the section below. </box>

Memory leaks

Facts noted by Olivier Salaün.

Constat

sympa.pl processes see their memory concumption brutally rise.

Characteristics:

- only sympa.pl is concerned
- the memory is never freed after this rise
- the memory rises correspond to large messages processing
- they happen even if the message is moved to bad (for example if it is too large)

Analysis tools

Methodology:

1. launche sympa.pl
2. `ps` auwx to know the size of the process (le figure to take into account is RSS, expressed in kB)
3. send a big message to a list
4. ps auwx efter the message was processed

Tools:

- /usr/local/sympa/src/tools/test-mem-leaks.pl: a small daemin which load a message at each loop. Usefull to understand which functions make the memory grow.
- /usr/local/sympa/src/patches/memwatch.patch: A patch to apply on Log.pm to add the momoery use at each log entry. It is possible to increase the log level and even add development traces to narrow on what part of the code is faulty.

Analysis

A test reveals the problem while sending a 26 MB message, rejected by sympa.pl:

1. process startup : 47MB
2. end of message processing : 1 273MB

The parts of the code in which memmory consumption rise (base=message de 26Mo) :

- +20Mo : Message::new / MIME::Parser::read()

<box blue|Remarks by Andras> I think it's normal, we can't do anything with it. </box>

- +52Mo : Message::new / MIME::Entity::as_string()

<box blue|Remarks by Andras> I replaced this one with read(FILE, $message→{'msg_as_string',-s FILE). This is saved 26Mb. </box>

- +26Mo : sympa::DoMessage / (parameter passing?)

<box blue|Remarks by Andras> I can't reproducate this. </box>

- +100Mo : sympa::DoMessage / do_log using the argument $message→{'msg_as_string'}

<box blue|Remarks by Andras> I hope this is only the part of development release and not the debugging. </box>

- +26Mo : sympa::DoMessage / creating a variable with $msg→as_string

<box blue|Remarks by Andras> We have to forbid to use $msg→as_string and force to use $msg→{'msg_as_string'} as reference! </box>

- +200Mo : sympa::DoMessage / call to tools::checkcommand()

<box blue|Remarks by Andras> This uses $msg→body which put the body of the message to an array. Every element of the array is one line of a message. This can't be efficient. </box>

- +80Mo : sympa::DoMessage / call to reject_report_msg()

<box blue|Remarks by Andras> We should never pass a message to the subrutine by value, only just by reference </box>

- +80Mo : List::send_file / dup_var

<box blue|Remarks by Andras> Similar to above: shouldn't copy a message object. Do you know Sympa why duplicate the message in this case? </box>

- +260Mo : mail::mail_file / TT2 parsing
- +170Mo : mail::mail_file / reformat_message
- +26Mo : mail::mail_file / passing a parameter to sending()

<box blue|Remarks by Andras> I hasn't checked these yet. </box>

- +50Mo : mail::sending / $msg→as_string

<box blue|Remarks by Andras> Same as above. </box>

Conclusions

To sum up, the memory consumption rise when:

1. a data structure is duplicated (dup_var ou $msg→as_string)
2. large data structures are passed as subroutines arguments.

<box blue|Remarks by Andras> …and the call tree is too deep, so perl GC can't reuse the same memory for the subrutines </box>

Possbile solutions:

- For (1) : increase the exploitation of the Message object; don't duplicate data, don't parse the same content several times.

<box blue|Remarks by Andras> Agreed. This is a hard work, I try to automatisate somehow. </box>

- For (2) : This problem seems to come from Perl. Deeper digging is required to solve it.

<box blue|Remarks by Andras> No, the solution is quite easy: pass a message to the subrutine as reference. The harder part is to find all the object which consists the body of the message. I'm searching for an debug utility which prints out those lexical objects which are bigger than a given size. </box>

<box blue|Remarks by Andras> The basic rules which we muss use to get rid of these duplications:

- If you create a new Message based on an earlier one, you should “overwrite” it, this saves a lot of works
- You shouldn't pass a message as string to a subroutine, the recommended way to pass the Message object as reference.
- You mustn't parse the body of the message as hash or array! These actions are very memory and CPU intensive.
- The code has to be refactorized, the big strings' name has to be signed. eg. big_body_as_string. This useful to determine which variables should be handled with care.
- If you create a new Message (eg. reject message) and the old Message object not used anymore it should be undefed.

</box>

From:
https://www.sympa.org/ - **Sympa mailing list server**

Permanent link:
https://www.sympa.org/contributors/incubator/sympa_performances

Last update: **2017/06/23 04:31**