The System Never Forgets. It Slows Down

Asko Tiidumaa[1], Erme Reino[2]

[1]Department of IT Services, University of Tartu, Estonia, asko.tiidumaa@ut.ee. [2]Department of IT Services, University of Tartu, Estonia, erme.reino@ut.ee.

Keywords
Web, information systems, response time, clickstream.

1. EXECUTIVE SUMMARY

1.1. Background

It takes about 7 seconds to lose a customer. That, on average, is the time a customer is willing to wait for a web page to appear. The universities, on the other hand, have the sole right to offer their services via electronic means. So there is no way to "vote with your wallet" by choosing another shop. The customer is desperate by being forced to the specific system and if waited long enough… abandons the request and tries to reload the page – either by submitting the data again or just hitting the reload button on the web browser. The server side, however, was probably overloaded, thus having negative impact on the response time. The system will continue to process those abandoned requests (zombies) and so the situation is eventually getting even worse.

Eventually, the customers may get frustrated and the system just might get back to normal again and one would never pay extra attention to such class of incidents. In the University of Tartu, however, we had a possibility to observe such a situation. Our study support system got severely overloaded and the amount of abandoned requests made up to one thirds from the total number of requests to the server.

1.2. Conclusions

We found that in our situation different problems appeared simultaneously, thus amplifying the negative user experience. It was not enough just to deal with the consequences, but the system was repaired using a sort of root cause analysis. We also propose a simulation method for diagnosing issues regarding to poor response times.

The main improvements were:

- Corrected application design and business logic;
- Optimizing the SQL statements;
- Changing the network topology between the nodes of the database cluster.

Last, regarding the topic of this paper:

- Encouraging the end users to be patient in times of extreme work load.

2. INTRODUCTION

This paper is to start a discussion about the phenomena of abandoned user requests and to give some methods for dealing with these. We will give a brief overview and analysis about our incident, in context of this phenomenon. We also propose methods to reduce the amount of abandoned requests and present the results from the corresponding experiments.

2.1. Our environment

Our study information system employs a 3-tiered architecture:

- Ordinary web browser;
- Web application server with a reports engine;
- Database server – a cluster with two nodes.

The application architecture is divided into three layers (presentation, business logic and database layer), which all run in the database server. The performance of the whole system is monitored using a vendor-provided monitoring and administration software. This gives us an especially good view over the database cluster and the real performance of queries it executes. The study information system has logs about every request made, with the complete parameter set and some timing information. There is also a possibility for a real-time view about current requests or actions.

3. THE INCIDENT

The 15-th of December 2007 was the first day students were allowed to subscribe to the topics of the next term. Before doing so, they had to fill out questionnaires about the topics taken earlier. If there were even one questionnaire unanswered, the student was not to let to subscribe to the new topics. After the midnight, the system serviced requests for about half an hour, freezing after that, despite the planned failover capabilities of a redundant database cluster. The services were restored some nine hours later. However, the response time remained extremely poor.

While analyzing the situation, the following observations were made:

1. The network topology of the database cluster interconnect was different than suggested by the best practices;
2. Most of the poor performance of the application was accounted to the topic subscription task;
3. The number of executions for SQL statement that checks whether the student has questionnaires unanswered, was enormously large;
4. Most of the poor performance of the database was accounted to one single SQL statement;
5. When the user did not get the response from the system soon enough, a reload button or a new action was executed within the user environment.

The first observation was made when the senior database administrator had to revive the system. Best practices of the vendor for setting up database clusters required, that nodes of a cluster should be interconnected through a separate network switch. Our database cluster had only two nodes so these were linked together by a direct network cable. This, however, rendered the node failover capabilities useless.

The second observation was self-explanatory, as the subscription season has just begun. At this time it was not clear, why it was taking so much time to complete the subscription requests. The clue lied in the third observation. The usual practice for topics subscription is that the student uses a search engine for finding the topics of interest, and picks the interesting ones out from the list returned. This list of results could be sometimes tens or hundreds of subjects long. It turned out that the check for unanswered questionnaires was performed not only once for the student while entering the subscription page, but for the every list item returned for the student to choose from. So, instead of executing the check just once, it was carried out sometimes even 100 times for a page view.

As this was just not enough, the SQL query that was used to check for questionnaires unanswered, was just written and not optimized, thus wasting the resources of the database server.

The last of this list of observations was addressing the existence of abandoned user requests. This means that the users have requested something from the system but have given up for the results to appear, and just pressed another button, or requested the web page to reload. This causes unnecessary workload on the database server, as it continues to process all the requests, and there is no possibility to cancel these for which there are no users waiting anymore. So, the system did not "forget" the request, it just slowed down. In our particular case, these abandoned requests made up to one thirds of the all requests made, as seen on Figure 1.

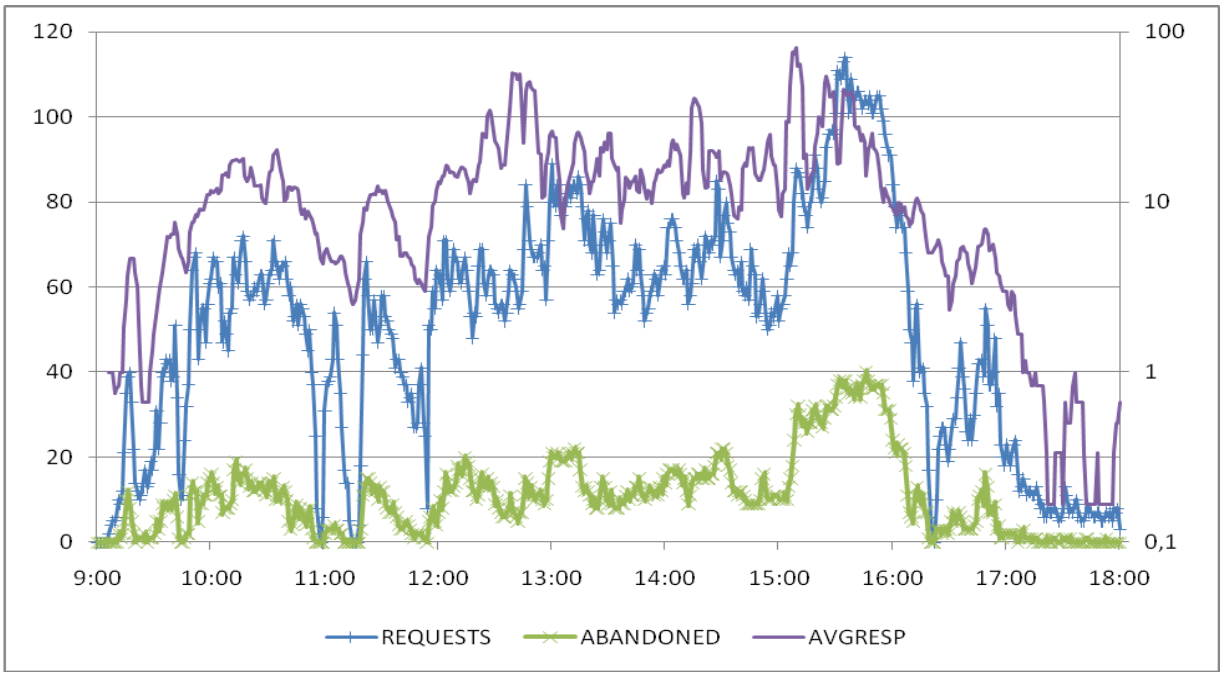


Figure 1. A performance overview about the incident

The goal was to make the system response time to a reasonable level. For this purpose, we analyzed the observations made during the incident. As the problems with proper application design and query tuning have been widely addressed, we do not deal with the specifics in this paper. However, as we have found no strategies for dealing with abandoned requests, we did research this topic a bit further and some experimentation was made to get some insight into the effectiveness of one of our proposed methods.

4. DEALING WITH ABANDONED REQUESTS

The existence of abandoned requests is rarely a problem of its own. An assumption was made that these appear due to the problems with poor response times of the application itself. We hereby outline some of the methods that could be used to handle or avoid this kind of a problem:

1. Kill the abandoned request, if a new action is requested;
2. Ignore the new action, and display the result of the abandoned request;
3. Delay the new action, until the abandoned request finishes;
4. Inform the user about temporal problems with system performance, asking for cooperation.

The first method, kill the abandoned request, looks promising but most probably needs technical support from the middleware or the database server. It is also necessary that the previous request can be positively identified, and to be sure that it can be discarded without any negative side effects. As web-based information systems are often stateless in nature and communication with the database in a well-designed application is done in a transactional manner, this should be taken as a positive assumption.

There is also a possibility to ignore the new action and display the result of the abandoned one. This could be most effective if the middle tier has a cache that stores the result of the action, which in our case is just a web page. However, this could be misleading for the end user, as the result may look something very different of what he or she has actually requested. The more time has passed between these two requests the more misguiding it can look.

The delaying of the new request may look somewhat simpler, although we see one big downside of this method – the waiting for the abandoned request to finish may become abandoned by itself, thus not being so effective. One can also make the waiting process for the end user more comfortable by showing a corresponding warning text.

This leads us to the last option, which is just displaying a warning to the end user that the system is having difficulties right now, and the response time may be a bit slower than anticipated. For technical point of view, this approach seems to be the most modest in context of special requirements for the middleware or the database server software. We only need to identify the current work load or average response time for the application, decide whether it is tolerable in context of abandoned requests, and display a warning text, if the anticipated response time exceeds some pre-defined threshold.

As in our study information system we have already a method for displaying custom warnings and errors, also a detailed log with response times, we opted to experiment with the last, cooperative method.

5. EXPERIMENTAL RESULTS WITH THE COOPERATIVE METHOD

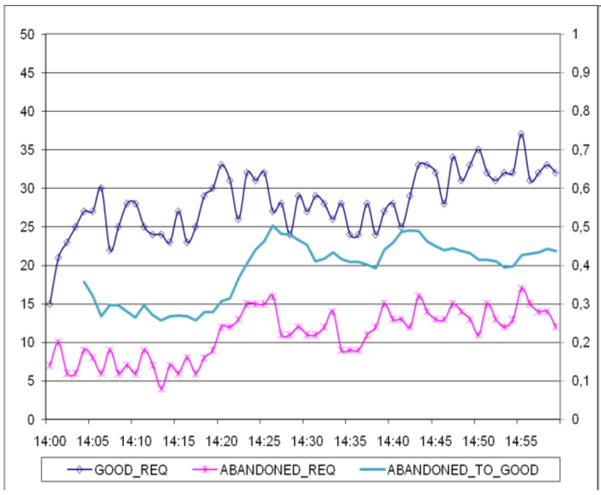
The implementation and integration of the cooperative method into our study information system was pretty straightforward. However, we had to make sure that it really works, as expected. The main question is: do end users really are so cooperative, and are more willing to wait for the result to appear?

This could be done by stress-testing the system. There are several methods for doing so. However, we did not have any idea of how to test this automatically, nor did we have any extra resources for a full blown stress-test. We opted to probe this assumption on our live study information system, using the real everyday human users. To do this, the system had to be taken into extremes in order to be slowed down. For this purpose, we chose to slow the system down by just using a simple delay in the application code.

The testing was conducted for a one hour period on a peak time, 2pm to 3pm, with each of this period having comparable predicted amount of requests per hour. To add more realism to the delay, we used a simple linear function $t'=at+b$ to determine the amount of time needed to wait. In this function, "t" represents the original response time for the specific action and "a" is a multiplier for this and "b" is just a constant for the delay. As we were satisfied with just some insight to the user behaviour, we conducted the tests with values "1" for "a" and "21" for "b".

On the first phase we made several runs without the warning text to get a "before" picture for the users' behaviour. However, it turned out that this kind of artificial stressing revealed several other problems, especially locking issues due to the missing indexes for foreign key constraints. These needed to be corrected before the tests could be continued. Figure 2 shows the one-hour period with a slow system, without the warning text. We can look that the data is comparable to that of the actual incident shown on Figure 1.

On the second phase, the displaying of the warning sign was enabled. The data collected of the one testing period is depicted on the figure 3. The comparision of the ratios of the abandoned vs good clicks gives us the encouragement to conclude our tests with an estimation that the end user is willing to cooperate.



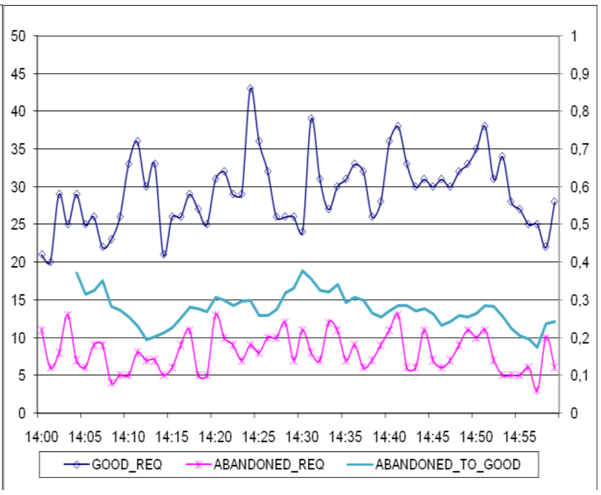


Figure 2. Phase 1 results **Figure 3. Phase 2 results**

It is also interesting to mention that while running the second phase tests we got positive feedback from the end users stating that it is much easier to wait for the results of a click when the system is apologizing for exceptionally long response times and begging for patience. This kind of feedback we do not see very often.

6. CONCLUSIONS

Besides reporting the lessons learned during resolving a particular problem, we must emphasize that sort of stress testing can be done quite easily by simply adding a delay to the application code. This makes the query response time artificially slower. In our case, it helped us to identify some locking issues in the database. We also got the courage to let the avoidance of abandoned requests left only to the cooperative method as the end users appeared to be supportive enough.

While the focus of this paper is dealing with abandoned requests, the most of the performance improvement came from redesigning the application, followed by SQL query optimization.

In contrast of Figure 1, the respective diagram for the next such a day would look rather boring – on average, every action is performed under one second, and on average there are 3..5 sessions active on each second. To achieve these results, the main improvements were:

1. Improved application design and business logic;
2. Identifying and optimizing the SQL statements. The resource consumption of the one particular query, that was used to check for unanswered questionnaires was reduced about 100 times;
3. Resolving locking conflicts that happened mostly due to missing indices that would optimize the enforcement of foreign key constraints;
4. Changing the network topology between the nodes of the database cluster. However, a better choice would be a cluster with at least three nodes;
5. Encouraging the end users to be patient in times of extreme work load.

To conclude our paper we must emphasize that the existence of the abandoned requests is not only a problem of its own but it is just a top of the iceberg. That means, that the users are repeating their actions due to existing problems with the application itself. The existence of the abandoned requests can be used as an indicator of problems related to the poor response times of the system. To get rid of these, the system must be fixed from top to down, starting with application design and business logic.

All else failing, we have presented you a last hope for retaining the trust of the end users – communicate with them, inform that the system is experiencing problems of temporary nature. They will be cooperative.

7. REFERENCES

Asko Tiidumaa. (2004). Knowledge Discovery in the Studies Information System of the University of Tartu, Proceedings of the 10th International Conference of European University Information Systems.

Oracle® Database Performance Tuning Guide, 10g Release 2 (10.2).

Oracle RAC Best Practices for Linux, From:
http://www.oracle.com/technology/products/database/clustering/pdf/twp_rac_bestpractices_10gr1_111403.pdf