Performance engineering as a case study

Margit Styrbæk Jørgensen

June 2016

Abstract

This report is an investigation of how to pinpoint performance problems in a specific case. Realising, investigating and addressing performance issues in the specific case should potentially yield experience to benefit future software development.

1 Motivation

When developing software systems one should, from an early stage, consider the performance of the system. It can however be necessary to address performance problems at a later stage.

Even in well-designed systems performance can become an issue for instance if the system is being used in another way than it was designed to, but it can also be the case that the system gradually reveals performance problems as it is being used, because realistic use cases has not been taken into account during test and development of the system.

In my organization we have an example of the latter: A web based system performing poorly because of bad design (I am allowed to say so as I am one of the developers).

It would be very interesting for me to let this report be an investigation of the performance problems and of the possible solutions to them. I would like to use tools and principles from my diploma courses, and I hope to gain experience to be able to pinpoint, analyse and address similar problems in other systems. I hope also to become wiser on knowing what not to do in future projects. All these outcomes would likewise be a great benefit to my organisation.

I am employed in a research department concerning agricultural science. The system in question is a Decision Support System aimed at agricultural advisors. The overall purpose of the system is to predict nutrient flow and greenhouse gas emissions by model simulations with the model: AnimalChange.exe

The web based interface acts as a user friendly way of generating input to the model by gathering user input about e.g. animals, feeding plans, crop rotations, soil types and fertilization strategy. These informations are saved as files and then fed to the model, which in turn returns result in the form of physical files saved on the webserver.
2 Hypothesis/Problem statement

The AnimalChange system is used by scientists to run scenarios and obtain model output. One of the Use Case scenarios is to adjust input, run model, assess output and, on the basis of this assessment, re-adjust input, re-run model a.s.o.

The model: AnimalChange.exe is the calculating “engine” operating with input and calculated output consisting of physical files and it can be run as a stand-alone program on a PC, but in the AnimalChange application it is run on a webserver.

The model is used to investigate the flow of nutrients and greenhouse gas emissions in farm scenarios before and after changes in management practices that are aimed at compensating for negative effects of climate changes.

This is done by calculating a baseline scenario for comparison, and it means that the result-files from the baseline scenario are used as part of the input for the model calculation of another scenario, the so-called projection scenario.

In this context, a scenario is a snapshot of a certain constellation on the farm in terms of crop types, harvest methods, animal feeding and fertilization strategy.

In the AnimalChange web application all user-related information about crops and yields, animals and manure, soil types and fertilizers are stored in a database and manipulated through the web application.

This is a great help in terms of generating valid input to the model, but the problem is that the response time increases with the complexity of the user input, and since complexity is a condition for most real scenarios, this has to be addressed. Complexity in this context means: More than 15 crops. As an example the first round of sequences in the Use Case utilized in this paper, with 19 crops, takes around 4 minutes to complete. Afterwards the result is assessed and input then may need to be re-adjusted a.s.o.

The reason for the need to re-adjust is that changes in one part of the system may affect other parts. For example increasing the amount of manure applied to a crop will increase the yield of the crop.

It is expected that the model calculations are slow because they are simulating 200 years of nutrient flow in baseline calculations and 20 years of flow in projection calculations, but the rest of the application has performance issues too.

![Fig.1 Deployment](image-url)
The performance problems can be connected to either CPU, memory, Network, Database or Disk read/write on the servers, or it can be connected to the application or to the model.

It is not my intention to perform any optimization on the AnimalChange model itself, only to find the performance problems of the supporting web page system.

My hypothesis is therefore:

It is possible to determine the most prominent performance bottlenecks in the AnimalChange web application, and – by solving them – to gain at least 10% performance.

3 Method

To investigate what is causing the performance problems, I needed a way to quantify and log the performance of the web and database server. For this I have used the Windows Performance Monitor as suggested by Tom Carpenter [1].

The application is not heavily used. It is not experiencing many users at the same time, but few users for longer periods each. To be able to measure performance on the specific application, I have created the same setup on a local computer: An Internet Information Server running only the AnimalChange application and connecting to the same database server.

I have installed a local database server with a copy of the database (SQLServer Express), This allows for testing with and without network involved in the data fetching.

In order to pinpoint the performance bottlenecks I have found inspiration in the catalogue of performance antipatterns (Smith et al. [2]), and in the two articles about optimizing data access in SQL Server (Al-Farooque [3][4]).

I have also made use of a capture/replay tool to simulate user transactions. This is a small keystroke recording/playback program called TinyTask.

One advantage of using a capture/replay tool is that user behaviour can be recorded and used as a constant input allowing comparable before/after performance measurements while changing various other parts of the system. Another advantage is that the tool can be scheduled and run repeatedly in a loop.

Capture/replay tools could also be used to build load generators for testing performance under heavy work load, but this will not be relevant in this investigation.

In order to measure time consumption specifically on the web pages, I have introduced a time measurement probe in the form of a JavaScript inserting timestamps in the browsers webStorage.

To measure time consumption specifically on the AnimalChange model I have utilized that it can be run as a stand-alone program, and I have recorded timestamps before and after call to the model.
4 Analyses and Results

The AnimalChange model takes input-files in the form of xml, and to help generating these files, the AnimalChange website have been developed.

The web system consists of a series of tab pages, each dealing with a specific area of the farming system. The relevant pages and areas are the following:

**Farm tab:** Several farm specific registrations are handled here, e.g. farming type and regional climate for the farm. On this page the model can be called to calculate results for the selected farm scenario.

**Yield tab:** Information about the amount of dry matter produced by the crops.

**Ruminants tab:** Information about the number and type of livestock, their feed plan, and housing.

**Manure tab:** Registration of amount and type of manure and fertilizer applied to the crops.

Additionally some tab pages are designed for presentation of data, and they are relevant for assessing the consequences of the input given on the former tab pages. These are the **Result tabs** and the **Balance tab**

![Web page screendump](image)

**Fig. 2 Web page screendump**

4.1 Capture/Replay use case
In order to investigate where performance problems are most prominent, a critical use case has been created. I have chosen this use case because it
represents a realistic farm, it makes use of the critical parts of the system and it demonstrates very long response times.

![Fig.3 Use Case overview](image)

**Capture/Replay use case - Manure distribution change:**

**Farm tab:** The farm scenario, that is set up to be the baseline scenario, is selected and calculated. In this step the web application is generating an xml input file containing the information for the scenario. The file is saved to the disk. The model is called and is running "outside" the web application. When finished, the model writes output to the disk.

**Result tab:**
When the model finishes, the result tab reads the model output-files from the disk and presents the result.

**Farm tab:** The projection scenario is selected in order to adjust and calculate.

**Manure tab:** The amount of a specific type of applied manure is changed and the changes are saved. This step in the use case is chosen to invoke the save procedure, and the specific amount of saved manure is not important.

**Manure tab:** After saving the manure tab is reloaded.

**Farm tab:** The model is called. Again the web application is generating input-xml, saves it to the disk. The model runs "outside" the application and writes output files to the disk.

**Result tab:** When the model finishes, the result tab reads the model output-files from the disk and presents the result.

**Balance tab:** The resulting balance is inspected

In this use case the model calculations are called two times: First to calculate the baseline scenario, and secondly to calculate the projection scenario. The baseline calculations are the most extensive simulating 200 years of nutrient flow, and the projection scenario takes the baseline result as part of the input and simulates 20 years of nutrient flow.
4.2 Performance monitoring of web and database server

The first question is: Are the web server and database server under stress? To get a picture of the overall state of the servers I have used Windows Performance Monitor on both servers.

To be able to log the various performance counters during operation of the application, I have made use of the capture/replay tool by running the Use Case scenario in a loop, and in that way letting it "hammer away" on the webserver and database server.

The logging is done this way because earlier recordings of “normal” use through the working hours of a week show that the average use of the web and database server is not heavy at all. This is due to the fact that the servers only hold small applications/prototypes with few user visits.

The loggings, with the use case loop, have been carried out during a timespan of 8 hours, from 23:20 - 07:20 and 10 minutes after logging-start the capture/replay loop was started, to simulate continuous application use. Around 07:10 the capture/replay loop was stopped.

The performance logs from this measurement yield the following graphs about CPU load, memory, hard disk utilization and network performance.

![WEB Processor (Total) % Processor Time](image1)
![SQL Processor (Total) % Processor Time](image2)

*Fig. 4 Web and SQL Server CPU performance*
According to Tom Carpenter [1] this CPU performance counter should be evaluated this way: If “Average % Processor Time” is less than 65%, there should be no reason to worry about the CPU workload. As for the database server this is OK. The monitored average is less than 25% on the Server. The “spike” on the SQL server diagram occurs at 02:03 which coincides with the start of the database backup job.

As it can be seen. The web server shows extensive CPU load during the repeated capture/replay calls. This is definitely something that has to be addressed. Tom Carpenter [1] suggests a high CPU utilization might be a symptom of memory problems and not necessarily of insufficient CPU speed. If the system is using an excessive amount of virtual memory, then this could account for the high CPU load.

The performance counter for the virtual memory is the “Memory Pages/sec” counter. This counter tracks the number of virtual memory pages read or written per second.

![WEB\Memory\Pages/sec](image1)

![SQL\Memory\Pages/sec](image2)

*Fig.5 Web and SQLServer Virtual Memory performance*

The average committed value concerning the web server is at least 400 during the capture/replay calls. It is hard to say if 400-500 Pages/sec represents an “excessive amount of virtual memory”, but it looks suspicious.

As for the Database server, the Memory Pages/sec is on average less than 5-10 and there is no reason to worry about this.
The hard disk performance is measured by the “Avg. Disk Queue Length” counter, and as long as the average queue isn’t exceeding 2, then there is no reason for concern. This is the case for the web server and for the database server as well. The spikes in the diagram occur at 02:00 on the web server and has apparently nothing to do with the call to the application. On the database server it occurs at 02:03 coinciding with the database backup job.

The last performance measure is the network performance counter for the two servers. If the average Output Queue Length is below 2 then the network device should be OK.
Fig. 7 Web and SQL Server Network Interface performance

The network interface is clearly not a problem in any of the servers, since the average Output Queue Length is around zero.

4.2.1 Conclusion of the server performance monitoring

There is clearly a problem with the CPU and/or the memory of the web server when operating the extensive Use Case scenario. I believe an upgrade of either RAM or CPU will have an impact on the performance of the AnimalChange application. Apart from this the network and disk read/write is OK on web server and database server: There is hardly no congestion concerning network traffic or hard disk read and write.

4.2.2 Web server upgrade

After an upgrade of the web server, the performance has been measured again. The same experiment has been carried out to test especially the CPU performance on the web server while operating the AnimalChange application. The test was this time only for half an hour, but the picture is quite clear.
The web server has been upgraded from 2 to 8 processors, and now the performance has improved. The logging started at 15:25 and the tiny task loop was started at 15:35. The larger spikes represent the baseline scenario being calculated and the smaller, last ones represent the projection scenario being calculated.

Before the upgrade, CPU was averaging almost 100% when the AnimalChange application was operated, and now the same experiment leads to no more than 30% Processor time. Both web server and database server should be able to cope with the amount of traffic on the AnimalChange application.

4.3. Setup on local computer

The memory/CPU on the web server is not the only problem. Database querying is also an issue. There is a time aspect in running the AnimalChange application connected to a local database or connected to a remote database, and this is not just important for the rendering of the web pages but also for the preparation of the input files to run the model. I conclude this because I experience the same response times in a setup on my local PC.

I have installed an Internet Information Server running only the AnimalChange application and a local database server with a copy of the database. This allows me to test the difference between running the application with or without having to query the database through network.

The model calculations are obviously the major time consuming part of the application, but the web pages providing the overview, editing features and input file generation for the model, are also slow, and by measuring loading time of the different pages in different parts of the use case, I will find out where to go into details.

All following measurements have been carried out on my local PC as it is “mirroring” the production environment.
4.3.1 Time measurement probe

To be able to actually measure the time consumption for rendering the separate ASP generated HTML pages, I need a probe. For this purpose I have created a JavaScript used for inserting timestamps in the start and end of a HTML page. The timestamp recordings should preferably not put additional load on the existing database system, and therefore the browsers web storage is used instead.

The probe is supposed to allow for recording of the timesteps in the use case

- **Capture/Replay use case - Manure distribution change:**

1. **Farm tab:** The baseline scenario is selected
2. **Farm tab:** The baseline scenario is calculated
3. **Result tab:** Result is automatically presented when ready
4. **Farm tab:** The projection scenario is selected
5. **Farm tab:** The Farm tab is reloaded
6. **Manure tab:** Manure applications are changed and saved
7. **Manure tab:** After saving the manure tab is reloaded
8. **Farm tab:** The projection scenario is calculated
9. **Result tab:** Result is automatically presented when ready
10. **Balance tab:** The resulting balance is inspected

*Fig. 9 Use case sequence measured*
// The 'count' function constructs a key name (load-counter) based on the 'page'
// parameter, and increments the number of times this specific load-counter
// has been set
// Values are stored in the browser’s sessionStorage and will be cleared
// on browser close down
// Example: count('Farm') -> key name = 'loadcount_Farm'

// The 'setLocalStorageParam' function constructs a key name based on the 'page'
// parameter, the related load-counter and the start_stop parameter
// The 'count' function is called when the parameter 'start' is applied
// A timestamp containing the number of milliseconds since 1970/01/01 will be
// stored as value for this key name
// Values are stored in the browser’s localStorage and will persist
// on browser close down
// Example: setLocalStorageParam('Farm', 'start') -> key name = '1_Farm_start'
// or '2_Farm_start' a.s.o. depending on the value of the load-counter value

function count(page) {
  var loadcount = ('loadcount_' + page).toString();
  if (typeof (Storage) !== 'undefined') {
    if (window.sessionStorage.getItem(loadcount)) {
      window.sessionStorage.setItem(loadcount, Number(window.sessionStorage.getItem(loadcount)) + 1);
    } else {
      window.sessionStorage.setItem(loadcount, 1);
    }
  } else {
    alert('Sorry, your browser does not support web storage...');
  }
}

function setLocalStorageParam(page, start_stop) {
  var loadcount = ('loadcount_' + page).toString();
  var countval = window.sessionStorage.getItem(loadcount);
  if (typeof (Storage) !== 'undefined') {
    var d = new Date();
    if (start_stop === 'start') {
      count(page);
      countval = window.sessionStorage.getItem(loadcount);
      var paramStartName = countval + '_' + page + '_start';
      window.localStorage.setItem(paramStartName, d.getTime());
    } else {
      var paramStopName = countval + '_' + page + '_stop';
      window.localStorage.setItem(paramStopName, d.getTime());
    }
  } else {
    alert('Sorry, your browser does not support web storage...');
  }
}
### 4.4 Time measurements

The probe is used in the following way: In the start and end of every tab page in the system the probe-script is called to create a key name/value pair. The key name is based on the name of the tab page and on the number of times the page has already been loaded. The value is a timestamp. In this way time measurements can be recorded while running the capture/replay use case under certain conditions.

The conditions are running the replay 100 times on the system with remote database access and afterwards 100 times with local database access. After each run the timestamps are fetched from the browsers localStorage and sampled into a spreadsheet. This produces the following averaged measurements.

<table>
<thead>
<tr>
<th>Sequence of tab pages loaded in the capture/replay</th>
<th>Avg. seconds from start to stop pr. page</th>
<th>Avg. seconds for model call to complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>1_Farm_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1_Farm_stop</td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td>2_Farm_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2_Farm_stop</td>
<td>0.003</td>
<td></td>
</tr>
<tr>
<td>1_ResultN_start</td>
<td></td>
<td>91.592</td>
</tr>
<tr>
<td>1_ResultN_stop</td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>3_Farm_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3_Farm_stop</td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td>4_Farm_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4_Farm_stop</td>
<td>0.006</td>
<td></td>
</tr>
<tr>
<td>1_Manure_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1_Manure_stop</td>
<td>0.020</td>
<td></td>
</tr>
<tr>
<td>2_Manure_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2_Manure_stop</td>
<td>0.020</td>
<td></td>
</tr>
<tr>
<td>5_Farm_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5_Farm_stop</td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td>2_ResultN_start</td>
<td></td>
<td>59.675</td>
</tr>
<tr>
<td>2_ResultN_stop</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>1_Balance_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1_Balance_stop</td>
<td>0.002</td>
<td></td>
</tr>
</tbody>
</table>
The column “Avg. seconds from start to stop pr. page” indicates the time consumption used to render the HTML page from ASP – from the time the HTML start to the HTML stops.

The column “Avg. seconds for model call to complete” indicates the time consumption from the loading-stop of the Farm tab to the loading-start of the Result tab. At these points the model calculations are called in the use case scenario.

It is clear that the page-rendering with local database connection is faster than with remote database.

The Result-pages are in all cases generated by reading physical files on the disk and therefore the database is not involved and it should not have any impact if the database is local or remote.

Even though the model is running on the local PC in both cases, the preparation of the input xml is depending on queries to the database and this accounts for the differences in time used for the model calls with remote or local database access.

### Fig. 12 Loading time difference

Apart from the Result pages, the rest of the time differences suggest some performance issues lie in the transport to and from the remote database.

If querying the database is an issue then several fixes can be applied e.g. the use of Stored Procedures, to minimizing queries to only fetch what is needed, and to make sure indexing is applied correctly.

However, the most prominent bottleneck is the model calculations, and to investigate the amount of time used in the actual model calculations, I have
performed 25 model runs with the use case input files through the AnimalChange model in a stand-alone setup. This experiment shows an average runtime of 13 seconds, but on the website the same calculation lasts approximately 1½ minute. This means the calculation and presentation section of the web pages should be looked into.

4.5 Calculation and presentation section of webpage

The principle in the calculation and presentation when performed on the web system is the following:

- The farm data is requested from the database
- Input file for the model is created from the farm data
- The model is called. It is reading input, calculating result and saving output on the disk
- The application waits for the output to be ready
- When output is ready, the file is traversed and output saved to the database
- On output presentation, the file is traversed

On the step where output is saved by traversing the output file, some waiting time has been necessary to prevent simultaneous reading and writing to the file. This waiting time is due to the fact that the file does not seem to be readable in the exact same time as it is registered as being present: The file is detected on the disk, but the AnimalChange model is not done writing to it.
After having detected the output file as being present, the system traverses it to save parts of the result to the database, and afterwards the result is presented. These tasks might as well be carried out at the same time and performed within the presentation procedure, and my refactoring is to move the saving procedure. It is not exactly saving any time because I just switch the order of execution, but

Moving the saving procedure means, in this case, moving code from ASP to C#.

*Fig. 14 Calculation and presentation sequence refactored*

I will also take a closer look at the waiting procedure. This is potentially unnecessary processing, one of the antipatterns listed by Smith [2]. In this case it is always at least 40 seconds no matter the actual running time of the model.

The functionality of the waiting procedure is to wait 40 seconds before opening the file. The 40 sec. of waiting ensures the file will be ready for reading in the worst case scenario, but at the same time it forces at least 40 seconds of waiting on every other case. To make the waiting procedure more scalable it is better to check the size of the file, because when it is still being written, the size will grow. My refactoring is to let the waiting procedure check for the filesize of the outputXML after having checked for the existence of this file.
Fig. 15 Wait procedure before and after refactoring

```plaintext
..
timel=now
tempSize0=-1
tempSize1=0
wait=5

while (tempSize0<tempSize1)
    tempSize0=objXMLDOM.GetFile("outputFile.xml").Size
    while (abs(datediff("s",now,timel)) <wait)
        timel=now
        tempSize0=objXMLDOM.GetFile("outputFile.xml").Size
   wend
    tempSize1=objXMLDOM.GetFile("outputFile.xml").Size
'***guard ensuring the loop will run at most 40 seconds***'
    if abs(datediff("s",now,timel)) <40 then
        if tempSize0=0 then tempSize0=-1
        else
            tempSize0=0
            tempSize1=0
        end if
        wend
..```

Fig. 16 Wait procedure - fileSize loop

4.5.1 Conclusion on refactoring of calculation and presentation section

The refactoring of this section of the application have had quite an impact on the performance of the calculation, and 100 runs with the use case capture/replay scenario and the probe shows this tendency. The experiment is done with local database access and the timing from the start of the model calculations to the start of the presentation page is recorded and compared with the 100 recordings from before the refactoring.
This is a big improvement because the major part of the long response times in the application is due to the model calls. The two model calculations are the specific ones used in the use case scenario. The first column (before) contains the model call timing measures from fig. 11.

### 4.6 Database optimization

The time measurements with the probe shows that the pages in the Capture/replay use case are rendered significantly faster when data to and from the database does not have to be transferred through network. This leads me to look into how network traffic to and from the database can be optimized.

One goal is to make the necessary queries perform as quickly as possible, another goal is minimizing the amount of data sent to, and received from the database.

Optimizing the database can be a relatively easy way to improve performance because the application code often will remain untouched. By the guidance from “Top 10 steps to optimize data access in SQL Server” (Al-Farooque [3][4]) the first task is to check the status of the database in terms of proper indexing. Indexes, especially primary keys enable the database engine to find the related records very fast because the data pages containing the table rows are physically sorted according to the primary key. When data are searched in a table missing a primary key, this forces the database engine to scan the entire table to find the corresponding row, which is a very slow operation. Depending on the amount of data in the table in question, this can therefore be a major reason for bad performance.

Another task recommended by Al-Farooque is to make sure the SQL code is implemented as Stored Procedures in the database and not as SQL in the application code. This is already the case in the AnimalChange application, and it allows for further refactoring of the Stored Procedures following some best practices (Al-Farooque [4]):

- Use “Set NoCount On” in Stored Procedures. This will prevent the message showing the number of rows affected by the query, from being sent as part of the message and thereby reduce network traffic

- Try to avoid dynamic SQL Dynamic SQL is generated on the fly, depending on parameter input, when the Stored Procedure is executed. The query execution plan of the dynamic SQL has to be calculated by the query optimizer every time the Stored Procedure executes

<table>
<thead>
<tr>
<th>Model call baseline</th>
<th>Avg. seconds for model call to complete (before)</th>
<th>Avg. seconds for model call to complete (after)</th>
<th>Model call after refactoring as percent of model call before</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model call projection</td>
<td>55.923</td>
<td>17.92</td>
<td>32.04%</td>
</tr>
</tbody>
</table>

**Fig.17 Model call response time before and after refactoring**
• Avoid selection of unnecessary data.
  Solving this task will decrease the network traffic and it will speed up the database performance
• Avoid 'SELECT *' in SQL queries.
• Avoid unnecessary columns in the SELECT list

4.6.1 Primary keys

The AnimalChange database contains 65 tables, and almost all tables have a composite primary key consisting of 2-4 columns, but I discovered that 9 tables was not indexed at all. My refactoring is to index these tables in the local and in the remote database.

4.6.2 Good practices in Stored Procedures

The AnimalChange database contains 104 Stored Procedures, and the refactoring is to let “Set NoCount On” be applied to all of them in the local and the remote database.

4.6.3 Refactoring of dynamic SQL

The AnimalChange application provides for user interaction in three languages. This means all static information strings are translated and stored in a language table. The Stored Procedure, for retrieving the language strings, makes use of dynamic SQL. This was originally thought to be the best solution since the number of different languages was not determined. However the chance of implementing additional languages is not high, and if new languages will be relevant in the future, it is not a big deal to edit a non dynamic Stored Procedure.

The dynamic SQL was replaced with plain, set based SQL, and this means the Stored Procedure will have to be altered if new language-strings are introduced in the database at some point in the future.

```sql
ALTER PROCEDURE [dbo].[GetLanguageStringNameList] @language char(2)
AS
BEGIN
SET NOCOUNT ON
DECLARE @strSQL varchar(200)
SET @strSQL = '
SELECT KeyWord,StringName_@language' + @language + ' FROM dbo.[Language]' exec(@strSQL)
END
GO
```

```
ALTER PROCEDURE [dbo].[GetLanguageStringNameList] @language char(2)='uk'
AS
BEGIN
SET NOCOUNT ON
IF @language='dk'
BEGIN
SELECT KeyWord,StringName_dk
FROM dbo.[Language]
END
IF @language='uk'
BEGIN
SELECT KeyWord,StringName_uk
FROM dbo.[Language]
END
IF @language='fr'
BEGIN
SELECT KeyWord,StringName_fr
FROM dbo.[Language]
END
END
```

Fig.18 Stored Procedure – ‘Dynamic’ to ‘set based’ SQL
4.6.4. Refactoring of unnecessary data

A single Stored procedure was refactored because it was using the “SELECT *” statement. The Stored Procedure: GetRotationList went down to only selecting half of the columns it did before. It was fetching 12 columns, but was only using 6 of them.

Additionally 7 Stored Procedures were using the “SELECT *” statement, but none of them were fetching unnecessary columns.

4.6.5 Conclusion - database optimization

After having made the changes to the database I registered 100 recordings with the probe. This time with the remote database connection to see if the refactoring of the database and of the calculation and presentation section have had an impact on the response times.

Remote database access before refactoring

<table>
<thead>
<tr>
<th>Sequence of tab pages loaded in the capture/replay</th>
<th>Avg. seconds from start to stop pr. page</th>
<th>Avg. seconds for model call to complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>1_Farm_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2_Farm_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1_ResultN_start</td>
<td>91.592</td>
<td></td>
</tr>
<tr>
<td>1_ResultN_stop</td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>3_Farm_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3_Farm_stop</td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td>4_Farm_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4_Farm_stop</td>
<td>0.006</td>
<td></td>
</tr>
<tr>
<td>1_Manure_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1_Manure_stop</td>
<td>0.020</td>
<td></td>
</tr>
<tr>
<td>2_Manure_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2_Manure_stop</td>
<td>0.020</td>
<td></td>
</tr>
<tr>
<td>5_Farm_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5_Farm_stop</td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td>2_ResultN_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2_ResultN_stop</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>1_Balance_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1_Balance_stop</td>
<td>0.002</td>
<td></td>
</tr>
</tbody>
</table>

Remote database access after refactoring

<table>
<thead>
<tr>
<th>Sequence of tab pages loaded in the capture/replay</th>
<th>Avg. seconds from start to stop pr. page</th>
<th>Avg. seconds for model call to complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>1_Farm_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2_Farm_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1_ResultN_start</td>
<td>17.562</td>
<td></td>
</tr>
<tr>
<td>1_ResultN_stop</td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>3_Farm_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3_Farm_stop</td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td>4_Farm_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4_Farm_stop</td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td>1_Manure_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1_Manure_stop</td>
<td>0.016</td>
<td></td>
</tr>
<tr>
<td>2_Manure_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2_Manure_stop</td>
<td>0.020</td>
<td></td>
</tr>
<tr>
<td>5_Farm_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5_Farm_stop</td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td>2_ResultN_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2_ResultN_stop</td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>1_Balance_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1_Balance_stop</td>
<td>0.002</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 19 Response times before and after refactoring
These tables do not show much effect of the database refactoring. An effect should show smaller response times, because the fetching from the database has diminished. Some response times went down some went up, but a sum of the response times gives 151 seconds before and 36 seconds after the two refactorings.

Many of the good practices are already used in the AnimalChange database e.g. using Stored Procedures as opposed to having the SQL in the code. This could be the reason that not much was gained by refactoring the database, but all these best practice advice will be used in other projects.

4.7 Model runtime with increasing number of items

It is clear that the AnimalChange model is using a lot of resources, and that it is the most prominent bottleneck in the system. It is expected and accepted, but the model is also performing slower with increasing number of crops.

To investigate the relationship between increasing number of crops and increasing model calculation time, I have created some control setups with different number of crops: 20, 40, 60 and 80 crops, and calculated these with the model 100 times.

The model is run as a stand-alone application and timestamps are recorded before and after each call to the model. The averaged figures are then plotted.

In this setup the input xml is already prepared and the run times are therefore not directly comparable with the run times in the use case scenario.

The resulting graph reveals that the relationship between increasing number of crops and model runtime is not absolutely linear. This should be investigated further. It is not my intention to optimize the model, but it would be a good idea to suggest this. The non-linear relation between the size of the input (the number of crops) and the model run time implies the search algorithms used in the
model is less efficient on larger datasets. This often passes unnoticed during test because test data are not large enough, and it could be one of the reasons why the overall performance of the AnimalChange system suffers when real (larger than expected) datasets are used on the model.

5 Conclusion

Performance engineering should be taken into account very early in the development of programs and applications, simply because it is easier to prevent performance problems than to solve them once the system is in production. This investigation was carried out to find solutions to the specific performance problems in the AnimalChange web application, and hopefully this experience will be an advantage to my future development tasks.

My experiments was first to examine if the web server or database server was under stress. This was done by the means of performance monitoring during use of the web application. These investigations showed CPU stress on the web server when being subjected to repeated model calculations through the web application, and the server was subsequently upgraded.

An important tool in the following experiments were the capture/replay tool TinyTask. This was used to create an automated use case operating the web application in order to record the part of the response times that lies between HTML start and HTML stop.

The recording were used to compare response times between use of the application with local or remote database access. They were also used to compare response times before and after refactoring, and especially the refactoring of the "calculation and presentation sector" in the web application had a big impact on the response times of the major bottleneck: The calculations.

After this refactoring response times of the calculations in my local setup went to 20-32% of the time that was used before.

Following some good practice advice I refactored the database to some degree. The major part of this was to apply "Set NoCount On" in all of the 104 Stored Procedures.

The model itself was not part of my plan for solving performance problems, but I conducted an experiment to find the extent of the growth in time consumption versus the number of crops in the input files. I found a non-linear relation and am going to bring this to the knowledge of the scientist and the programmer of the model.

My aim was to find the most prominent performance bottlenecks in the AnimalChange web application, and – by solving them – to gain at least 10% performance. This has been possible mainly due to the changes in the "calculation and presentation section", and in the use case experiment calling the application with remote database access after all refactorings (Fig. 19) I can conclude that it was possible to gain at least 10% performance improvement. In fact the improvement was larger. The improvement in response time with remote database access means the calculation response times are from 19-31% (Fig 19) of what they were before.
References


