Performance Tuning of an Intranet Application - A Case Study

Georg Sonneck, Thomas Mueck
Institute of Computer Science and Business Informatics
University of Vienna
Rathausstr. 19/9, 1010 Vienna, Austria
{sonneck,mueck}@ifs.univie.ac.at

Abstract
There is little doubt that performance issues play an important role in web applications. Only few situations are as boring as sitting at a computer facing a web browser and waiting for a (usually dynamically generated) web page to appear. This paper discusses important aspects of empirical performance evaluation and tuning, which can be a rather complex task because of the standard structure of web-applications - usually there is more than one server involved. To evaluate the performance of the application empirically, the reply time of request and internal performance measurement techniques are used.

In the context of a real life intranet information system, we elaborate both, on evaluation and tuning of web server as well as database server performance.

As a result of the performance tuning steps presented in this paper we reduced the turn around reply time for typical pages of the tuned application from approximately 300 seconds to 80 seconds on average.

1. Introduction
Evaluating the performance of web applications has already been subject of several research projects and similar investigations, e.g., [1], [4] and [9]. This paper proposes an attempt to combine empirical performance evaluation with tuning steps. From our point of view, the performance tuning of web based information systems is rather challenging because of the heterogeneity of server components that have to work together in an optimal way in a multi tier architecture.

The paper is structured as follows: This Section summarises the tuning approach, related work as well as the application requirements. This is followed by an overview of the application architecture in Section 2. In the third Section a model for performance evaluation and tuning is proposed. The fourth Section identifies stress test scenarios and discusses the relevant parameters for tuning. Section 5 describes commercial stress test tools and our approach, which had to be chosen because of specific application demands. In this Section the special functionalities of the application and the new parts of the stress test tool are covered. The next two Sections show the results of the initial stress test, necessary tuning steps and the final performance results of the application. They are followed by the conclusions.

1.1. An Approach Towards Efficient Performance Tuning

In this paper performance tuning is viewed as an iterative process during the whole development of an application. The opinion that performance tuning is the last step of software development is considered wrong. Some performance decisions, for example the creation of clusters, have to be made during the design phase of the database. Nevertheless it is also important to tune a system at the end of the software development process. Only these final stage tuning activities are covered in this paper.

Tuning is impossible without knowledge of the initial performance of the system, which is measured using stress tests. We feel that it is more efficient to tune only one server at a time to see the performance gains of isolated components more clearly. Tuning two or more servers simultaneously may conceal performance losses of one component, if the other components perform better.

1.2. Related Work

Performance analysis of web based information systems definitely is a hot topic at the moment. For example, in [9] a standard workload and performance model in the realm of stochastic queuing theory is applied thus utilising prior work described in [6]. However, analysis in close conjunction with tuning scenarios is less frequent. In this paper we
use a systematic empirical black box approach with stress test tools.

1.3. Application Requirements of the Web Based Information System

The main goal of the project described in this paper was the tuning of an information system for financial advisors who have to maintain and control relevant customer information, in particular insurance contracts, income, marital status, birthday, and so on. This information system includes the production of a supporting document for the marketing process in the form of a PDF document, describing the financial situation of the customer in various fields of interest, based on previously inserted customer data.

As of 01/10/2001 the application has to deal with the following data volumes

- Customers: 120.994
- Contracts concerning customers: 304.501
- other firms or institutes: 1040
- Answers of additional questions from customers: 2.396.788

Approximately 1000 employees (financial advisors) use the system. Employees can use the system in a 24*7 uptime setting. Under these circumstances we estimate a maximum number of 50 employees using the system simultaneously, i.e., tuning activities are done for up to 50 concurrent users.

2. Application Architecture

As figure 1 shows the system is an intranet information system, consequently only employees are allowed to use it. They can connect to the apache web-server either via the company internal intranet or using the internet utilising a standard firewall and a secure connection (https). This was done to make life easier for external agents. The user interface is implemented by dynamic perl-scripts. These scripts are called dynamic, because they present information about customers that are stored in a relational database in a dynamic way.

The oracle database does not only store the customer data, it also contains the stored procedures for a model driven financial analysis. This component performs an analysis of the financial situation of customers and produces a printable PDF-document as result. As will be shown in Section 6 this analysis is one of the most time-critical parts of the application. The whole calculation is done directly in the database using PL/SQL whereas the Web-Server provides mainly the front end to the user. In the Perl-Modules only the style information of the output pages is stored. Doing this the administration is simplified and the time needed to arrange the pages is minimized.

3. Model for Performance Tuning

The most important parameter concerning empirical performance tuning is the time delay of the reply. Because of this reason the stress test is based mainly on the evaluation of this parameter. Other parameters like memory or cpu usage of the server have been background information that became important only in specific situations which will be described in detail.

For the tuning process a model consisting of several phases was used:

- This analysis in the first phase is used to find out critical parts concerning performance. The analysis was
Table 1. Stress test scenarios

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Description</th>
<th>RF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management of customer data</td>
<td>Insertion, modification of core customer data and modeling of the relationships between customers, for example marriage</td>
<td>10%</td>
</tr>
<tr>
<td>Insertion of customer contracts</td>
<td>Customer financial data (in form of contracts and supplementary questions) is inserted</td>
<td>30%</td>
</tr>
<tr>
<td>Modification of customer contracts</td>
<td>Customer financial data as described above is modified</td>
<td>10%</td>
</tr>
<tr>
<td>Calculation of model driven financial analysis</td>
<td>After a check whether or not calculation is possible, the calculation is done and the PDF-File is generated</td>
<td>50%</td>
</tr>
</tbody>
</table>

done using stress tests. Before a stress test can be started one has to identify typical user sessions to make the test more realistic.

- In the second phase the Oracle DB-Server and the Apache Web-Server are tuned one at a time whereas the other stays in the original configuration. To proceed like this has the advantage to isolate component specific performance gains and losses from each other.

- The third phase starts when both servers have been sufficiently tuned. This phase consists of a stress test of the whole system. If the overall performance gain is sufficient the process ends, otherwise one returns to the second phase where the tuning steps are changed to produce a better result.

4. Stress Test Scenarios

Several usage profiles can be distinguished when using this application. Table 1 lists the most important use-cases representing typical user transactions. For each use-case a short description of the included transactions and a relative frequency of usage is noted. In that way the relative frequency specifies the importance of each use-case. Stress tests perform user-transactions belonging to a use-case according to the known relative frequency.

The application has been stress tested on the internet on different days at different times because the reply time is also dependent on the overall load of the communication lines.

5. Implementing Stress Tests

5.1. Commercial Stress Test Tools

The following commercial tools were analyzed:

**Benchmark Factory** [3]
This tool allows to stress test several software systems such as: Messaging Servers, Web-Servers, Databases. In our situation the only relevant part is the stress test of the Web-Server because the interaction of users with the system is implemented via WWW. Each stress test is done within a project. It is possible to specify the following elements:

- **Single User Part** equivalent to one HTTP-Request
- **User Sessions** have one or several Single User Parts
- **Multi User Parts** combine several User Sessions with relative frequencies.

Agents execute the Multi User Parts. It is possible to specify a "think time" i.e., a simulated response delay too. At the end the system is analyzed using several plots. The advantages of the software are that there exist libraries for reuse and that user sessions can be directly recorded using a built in browser.

Unfortunately, because of two reasons, it was not possible to use this tool for our application:

- The HTTP-Requests are static, that is to say the database queries will return the same rows each time.
- It is not possible to integrate the session-administration of the application.

**Webload** [8]
This stress test tool is very similar to the Benchmark Factory. It is also not possible to generate dynamic HTTP-Requests.

5.2. Development of an Application Specific Stress Test Tool

The problems with the commercial tools discussed above were solved by developing a new tool, which consists of the following components:

- The Request of a Page (RoAP) is identified by an URL.
- Use-cases, which are constructed of several RoAP’s.
- Sessions, which call a use-case several times according to the relative frequency of each use-case.
- Test-Sessions, which include m sessions according to the number of simultaneous clients. For each test-session a Log-File is generated that stores the different URL and the average reply time for each URL.
• General-Test, which starts several test-sessions with different numbers of simultaneous clients.

The tool implements also a time to reflect (in seconds) and a time-separation - this is the time between logins of consecutive clients. The problem of non static page requests because of the session management of the application is solved too.

6. Results of Initial Stress Test

To get an initial state of the performance the whole system was stress-tested before any tuning was performed.

Figure 3 shows the result for the two most critical dynamic pages of the application: The person search page, which is of central importance because an employee has to select a customer before he or she can do any other step and the calculation of the model driven financial analysis, which has been identified as one of the most often used use-cases.

![Figure 3. Performance before tuning.](image)

The following conclusions are obvious:

• The period of reply of the pages is very high from the beginning and

• increases very fast when the number of simultaneous clients grows.

7. Tuning Issues and Improvements

Here we are focusing only on Oracle-specific tuning [2] and the changing of Apache system-settings [5] - no change was made in the hardware of the system. The first SubSection gives a brief overview on the tuning activities for the Oracle server. The apache server tuning will be described in the second SubSection where we will show the tuning steps taking two parameters as examples for all the others. The whole tuning steps are described in more detail in a master thesis [7].

7.1. Oracle Database Tuning

To evaluate the performance Oracle internal features have been used. We will show Oracle tuning and performance evaluation on different levels. Firstly we will look at initialization parameters playing an important role concerning performance. These parameters are placed typically in the INITDatabasename.ora file. Some of these parameters, for example the DB_BLOCK_SIZE parameter, must not be changed after installation. Because of that it is important to choose an appropriate value for this parameter at a very early stage of the development.

The main duty of a part of the SGA (System Global Area) is caching. Its size can be calculated by multiplying the parameters DB_BLOCK_BUFFERS and DB_BLOCK_SIZE. In listing 1 we show a possible way to check if this part of the SGA is too small by using the Oracle internal view $sysstat.

```sql
SQL> select sum(Value) as L
2    from V$sysstat
3    where name in ('db block gets',
4      'consistent gets');
L
---------
5807

SQL> select sum(Value) as P
2    from V$sysstat
3    where name in 'physical reads';
P
---------
836

TQ = \frac{(L - P)}{L} \times 100 \tag{1}
```

Listing 1: Ratio of logical and physical reads

In Listing 1 L represents the sum of logical block access whereas P indicates the sum of physical reads. For our system the ratio TQ is 85.60% which is good enough.

Another parameter that is of importance concerning performance is the SHARED_POOL_SIZE, which defines the size of the global cache that is used by all processes. It is possible to mark specific objects with the use of Oracle commands to reside permanently in the Shared Pool. When we look at the hitratio for objects in our database we get the result shown in listing 2.

```sql
SQL> select namespace, gethitratio
2   from V$librarycache;
NAMESPACE GETHITRATIO
---------- -----------
SQL AREA .810810811
```

0-7695-1573-8/02/$17.00 (C) 2002 IEEE
Because of the result, the hit ratio is high enough, it is decided to leave the parameter at 60.000.000.

A second very important way of tuning the database side of an application is to tune critical SQL statements directly. A statement is decided as critical when it

- has a high response time, or
- is used very often.

To identify these statements one can use the view V$SQLAREA that has statistics for each statement that is present in the cache.

According to the first stress test (see Fig. 3) the Search-for-customer page and the model driven financial analysis have critical response times.

Listing 3 shows the critical SQL statement for the Search-for-customer page.

```
SELECT i.CU#, firstname, lastname,
Postal_Code, Street, p.P#
FROM Personphys p, Int i, Persongen pg
WHERE p.P#=i.P# AND pg.P#=p.P# AND
lastname LIKE 'GRAF%'
AND i.respemp IN
(SELECT emp# FROM Employee
WHERE STRUKT LIKE '%');
```

Listing 3: Search-for-customer SQL-statement

This query is getting so time-consuming because an employee is allowed to see all customers that belong to employees under his control too. The attribute respemp contains the number of the employee that is responsible for this customer. The STRUKT attribute of an employee consists of the STRUKT attribute of his leader plus his own employee number. In our case we assume that the employee doing this query has the rights to see everything inside the organisation and has consequently access to all customers of all employees.

To simulate a realistic situation the concerning tables (Personphys, Int, Persongen, Employee) are already filled with representative data. At the starting point the involved tables are not analyzed, do not belong to a cluster and have no additional indexes.

Listing 4 shows the execution plan for the statement.

```
SQL> explain plan for
SELECT i.CU#, firstname, lastname,
Postal_Code, Street, p.P#
FROM Personphys p, Int i, Persongen pg
WHERE p.P#=i.P# AND pg.P#=p.P# AND
lastname LIKE 'GRAF%'
AND i.respemp IN
(SELECT emp# FROM Employee
WHERE STRUKT LIKE '%');
```

Listing 4: Execution plan of Search-for-customer SQL-statement

The execution time for the query takes 10 seconds. Because of the empty column COST we can assume that the RULE-BASED-OPTIMIZER will be used. This optimizer is used when no statistics are available for the involved tables. The join between the tables is realized via a NESTED LOOP. For the table PERSONGEN a FULL TABLE SCAN is done. We would like to use the Cost-Based-Optimizer.

To achieve this goal we have to analyze the involved tables using the SQL statements shown in listing 5.

```
analyze table Employee compute statistics;
analyze table Persongen compute statistics;
analyze table Personphys compute statistics;
analyze table Int compute statistics;
```

Listing 5: Analyzing tables
Listing 6 shows the corresponding change in the execution plan. The Cost-Based-Optimizer is used now which can be seen on the non empty column COST.

<table>
<thead>
<tr>
<th>Query Plans</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT STATEMENT CHOOSE</td>
<td>6118</td>
</tr>
<tr>
<td>HASH JOIN</td>
<td>6118</td>
</tr>
<tr>
<td>TABLE ACCESS FULL EMPLOYEE ANALYZED</td>
<td>14</td>
</tr>
<tr>
<td>HASH JOIN</td>
<td>5613</td>
</tr>
<tr>
<td>HASH JOIN</td>
<td>3064</td>
</tr>
<tr>
<td>TABLE ACCESS FULL PERSONPHYS ANALYZED</td>
<td>193</td>
</tr>
<tr>
<td>TABLE ACCESS FULL PERSONGEN ANALYZED</td>
<td>220</td>
</tr>
<tr>
<td>TABLE ACCESS FULL INT ANALYZED</td>
<td>111</td>
</tr>
</tbody>
</table>

8 rows selected.

Listing 6: Execution plan with analyzed tables

Not satisfactory is however that a hash join is used to join the tables and that a full table scan is used to retrieve the corresponding data.

In creating Btree-indexes for the attributes lastname and respemp and one Bitmap-index for the column aktiv we try to improve the performance further. For the column aktiv a Bitmap-index is used because of the low cardinality - only the values ['J', 'N'] are accepted - of this column. Listing 7 shows the SQL-statements to create the described indexes.

```sql
SQL> create index i_n on personphys(lastname) tablespace indtab;
Index created.
SQL> create bitmap index i_a on persongen(aktiv) tablespace indtab;
Index created.
SQL> create index i_o on int(respemp) tablespace indtab;
Index created.
```

Listing 7: Creation of indexes

Listing 8 shows the corresponding execution plan. As one can see the estimated costs are reduced. Only the additionally generated index i_n is used. The table access is no longer a full table scan, but the indexes are used to retrieve the queried data.

<table>
<thead>
<tr>
<th>Query Plans</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT STATEMENT CHOOSE</td>
<td>23</td>
</tr>
<tr>
<td>NESTED LOOPS</td>
<td>15</td>
</tr>
<tr>
<td>TABLE ACCESS BY INDEX</td>
<td>11</td>
</tr>
<tr>
<td>ROWID PERSONPHYS ANALYZED</td>
<td>33</td>
</tr>
<tr>
<td>INDEX RANGE SCAN I_N</td>
<td>2</td>
</tr>
<tr>
<td>TABLE ACCESS BY INDEX</td>
<td>2</td>
</tr>
<tr>
<td>ROWID INT ANALYZED</td>
<td>1</td>
</tr>
<tr>
<td>INDEX UNIQUE SCAN S1460</td>
<td>1</td>
</tr>
<tr>
<td>TABLE ACCESS BY INDEX</td>
<td>1</td>
</tr>
<tr>
<td>ROWID PERSONGEN ANALYZED</td>
<td>2</td>
</tr>
<tr>
<td>INDEX UNIQUE SCAN S01434</td>
<td>1</td>
</tr>
</tbody>
</table>

Listing 8: Execution plan with supplementary indexes

Another way to increase performance is the detection of fragmented tables and the annulation of these fragments. The necessary steps are shown in listing 9 where the table Persongen is checked.

```sql
SQL> analyze table PERSONGEN
2 list chained rows
3 into chained_rows;
Table analyzed.
SQL> select count(*) from chained_rows;
COUNT(*)
----------
3896
```

Listing 9: Detection and annulation of chained rows in a table
7.2. Apache Webserver tuning

The father process of the web server starts a specified number of child processes that serve the incoming requests. The following enumeration lists important points concerning the tuning of the apache server: [5]

- The more processes are allowed the bigger can be the workload for the processor.
- The more processes are running the more memory is needed.
- The more processes are running the more resources are required.
- But more processes mean also that more requests can be served simultaneously without loss of time.

One has to find a compromise between two extremes. Here we discuss only the two most important parameters that influenced performance a lot in our system.

**Perl Handler Apache::Registry vs. Apache::PerlRun**

We tried to change the directive PerlHandler Apache::PerlRun to PerlHandler Apache::Registry. The results are shown in figure 4. PerlHandler Apache::Registry is obviously much faster than PerlRun. But it is remarkable that the reply time is nearly the same for one simultaneous client. With the directive PerlRun the result needs 1.15 seconds whereas Apache Registry needs 1.22 and is even slightly slower. The explanation of this phenomenon is that the apache server has been restarted before the start of a whole stress test. Apache Registry has to load the modules into memory and cannot be faster than PerlRun at the beginning.

**MaxRequestsPerChild**

MaxRequestsPerChild is the second parameter that we will describe in more detail. We started with a directive of 0, which means, that a generated apache child process can serve an infinite number of clients. In this case it is necessary, of course, to control the memory situation of the server to see if memory leaks exist leading to a lower overall performance of the application.

The memory situation before the stress test was:

<table>
<thead>
<tr>
<th>PID</th>
<th>TTY</th>
<th>SIZE</th>
<th>SWAP</th>
<th>RSS</th>
<th>SHRD</th>
<th>DT</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>3675</td>
<td>?</td>
<td>8712</td>
<td>0</td>
<td>8712</td>
<td>3692</td>
<td>142</td>
<td>/usr/sbin/httpd -f /etc/httpd/httpd.conf</td>
</tr>
<tr>
<td>3676</td>
<td>?</td>
<td>8712</td>
<td>0</td>
<td>8712</td>
<td>3692</td>
<td>142</td>
<td>/usr/sbin/httpd -f /etc/httpd/httpd.conf</td>
</tr>
<tr>
<td>3677</td>
<td>?</td>
<td>8712</td>
<td>0</td>
<td>8712</td>
<td>3692</td>
<td>142</td>
<td>/usr/sbin/httpd -f /etc/httpd/httpd.conf</td>
</tr>
<tr>
<td>3678</td>
<td>?</td>
<td>8712</td>
<td>0</td>
<td>8712</td>
<td>3692</td>
<td>142</td>
<td>/usr/sbin/httpd -f /etc/httpd/httpd.conf</td>
</tr>
<tr>
<td>3679</td>
<td>?</td>
<td>8712</td>
<td>0</td>
<td>8712</td>
<td>3692</td>
<td>142</td>
<td>/usr/sbin/httpd -f /etc/httpd/httpd.conf</td>
</tr>
</tbody>
</table>

Figure 5 shows that there existed ten apache processes with ten related oracle-sqlplus processes. Each oracle process had an approximate memory consumption of 15,5 MB. A restart of the apache server caused also the termination of the related oracle-sqlplus processes and in consequence normalized the situation, which led to 60888K free and 57752K shared memory.

Because of the increasing memory usage of oracle-sqlplus processes related to apache child processes it is better to force the termination of apache child processes after a certain number of fulfilled requests. Therefore we set the directive to 30 which results in a better balance between memory growth of related processes and the need to create new apache child processes too often.

Doing a stress test with up to 50 simultaneous clients resulted in the following situation:

Mem: 322932K av, 319784K used, 3148K free, 228944K shrd, 11776K buff

The server had nearly no free memory left. We used the command ps axwm to identify processes consuming a high amount of memory (see figure 5).
A general stress test was done after the servers had been tuned independently and resulted in the following situation:

\[\text{Figure 6. Performance after tuning.}\]

As the performance shown in figure 6 already satisfied the customer’s request, tuning was finished at this point.

8. Conclusions

The paper shows an empirical analysis and tuning approach in the context of a real life intranet application. Web server as well as database server performance are tackled in an isolated fashion. However, in this case study the isolated tuning steps of each subsystem also led to an overall performance gain. It will be the aim of further research to investigate under which circumstances the performance improvement of each isolated subsystem could result in a performance loss of the overall system and how to avoid such a drawback.

References