Case Study V:  
A Help-Desk Service

Prof. Daniel A. Menascé  
Department of Computer Science  
George Mason University  
www.cs.gmu.edu/faculty/menasce.html

Copyright Notice

• Most of the figures in this set of slides come from the book “Performance by Design: computer capacity planning by example,” by Menascé, Almeida, and Dowdy, Prentice Hall, 2004. It is strictly forbidden to copy, post on a Web site, or distribute electronically, in part or entirely, any of the slides in this file.
Capacity Planning for Existing Workloads

Existing Workload → Existing Workload → Existing Workload

Measurements on the Real System → measured service demands → Performance Model Solution → Performance Metrics

Capacity Planning for New Workloads

New Workload → Application Sizing

Existing Workload → Existing Workload → Existing Workload

Measurements on the Real System → measured service demands → Performance Model Solution → Performance Metrics

© 2004 D. A. Menascé. All Rights Reserved.
Software Performance Engineering (SPE)

• A set of methods and techniques to design software systems in such a way that they will adhere to performance requirements once implemented.
• SPE has to be applied at different stages during the life cycle of a software system so that estimates on services demands are continuously refined.
• Software Performance Engineers need to provide feedback to developers during the software development stage to improve the performance of the system in question.

The Help-Desk System

• A high-tech company has 21,600 employees with access to a variety of computing resources.
• A new help-desk application is being designed:
  – Creation of help tickets. If the FAQ DB does not solve the problem a help ticket is created.
  – Tracking and verification of the status of open help tickets.
Data for Application Sizing

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of employees</td>
<td>21,600</td>
</tr>
<tr>
<td>Percent employees that access FAQ DB per day</td>
<td>0.20</td>
</tr>
<tr>
<td>Percent access to FAQ DB at peak period</td>
<td>0.80</td>
</tr>
<tr>
<td>Avg. number of queries to FAQ DB per access to FAQ DB</td>
<td>4.00</td>
</tr>
<tr>
<td>Percent of accesses to the FAQ DB that generate ticket creation</td>
<td>0.35</td>
</tr>
<tr>
<td>Percent of employees that create new tickets/day without going to the FAQ database</td>
<td>0.15</td>
</tr>
<tr>
<td>Percent of new ticket creation at peak period</td>
<td>0.54</td>
</tr>
<tr>
<td>Percent of employees that inquire about new ticket status/day</td>
<td>0.10</td>
</tr>
<tr>
<td>Percent of status inquiries at the peak period</td>
<td>0.90</td>
</tr>
<tr>
<td>Peak period duration (in seconds)</td>
<td>7,200</td>
</tr>
<tr>
<td>Arrival rates during peak period (requests/sec)</td>
<td></td>
</tr>
<tr>
<td>FAQ queries</td>
<td>1.9200</td>
</tr>
<tr>
<td>New ticket creation</td>
<td>0.4110</td>
</tr>
<tr>
<td>Ticket status</td>
<td>0.2700</td>
</tr>
<tr>
<td>Period (in days) during which tickets are kept</td>
<td>365</td>
</tr>
</tbody>
</table>
Workload Intensities

\[ \lambda_{FAQ} = \frac{N_c \times f_{FAQ} \times P_{FAQ} \times N_q}{T} = \frac{21,600 \times 0.2 \times 0.8 \times 4}{7,200} = 1.92 \text{ requests/sec} \]

\[ \lambda_{ticket} = \frac{N_c \times [(f_{FAQ} \times P_{FAQ} \times l_{FAQ}) + (f_{t} \times P_{t})]}{T} = \frac{21,600 \times [(0.2 \times 0.8 \times 0.35) + (0.15 \times 0.54)]}{7,200} = 0.41 \text{ requests/sec} \]

\[ \lambda_{status} = \frac{N_c \times f_{t} \times P_{t}}{T} = \frac{21,600 \times 0.1 \times 0.9}{7,200} = 0.27 \text{ requests/sec} \]

Entity-Relationship Model for the Database Design

© 2004 D. A. Menascé. All Rights Reserved.
Database Design for the Help-Desk System

<table>
<thead>
<tr>
<th>Database Table</th>
<th>Cardinality</th>
<th>Row Size(bytes)</th>
<th>Total size(Kbytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question</td>
<td>10,000</td>
<td>1000</td>
<td>10,000</td>
</tr>
<tr>
<td>Keyword</td>
<td>500</td>
<td>50</td>
<td>25</td>
</tr>
<tr>
<td>Employee</td>
<td>21,600</td>
<td>140</td>
<td>3,024</td>
</tr>
<tr>
<td>Ticket</td>
<td>1,734,480</td>
<td>260</td>
<td>450,965</td>
</tr>
<tr>
<td>KeywordQuestion</td>
<td>100,000</td>
<td>20</td>
<td>2,000</td>
</tr>
<tr>
<td>TicketEmployee</td>
<td>1,734,480</td>
<td>20</td>
<td>34,690</td>
</tr>
<tr>
<td>TicketKeyword</td>
<td>8,672,400</td>
<td>20</td>
<td>173,448</td>
</tr>
</tbody>
</table>

Assumptions:
- A ticket is kept in the DB for one year.
- One question is associated with 10 keywords, on average.
- One ticket has 5 keywords associated to it, on average.
- Questions are 1000 bytes long.
- Keywords are 50 bytes long.
Specifying Transaction Logic for SPE Purposes

• Use a language that captures the major structural components of a transaction.
  – Loops and average number of times executed.
  – Branch statements and branching probabilities.
  – Switch statements and case probabilities.
  – Database access (i.e., select and update) statements.
• Estimate number of I/Os per transaction and estimated the CPU time from the number of I/Os using benchmark data.
• Example of such a language: Clisspe

Transaction Logic for Query on FAQ Database

01 loop #KeywordsPerQuery
02 ! obtain keyword id for a given a keyword
03 select from Keyword where Keyword;
04 ! obtain all question ids for a given keyword id
05 select from KeywordQuestion where KeywordId;
06 ! access the Question table to retrieve selected questions
07 loop #QuestionsPerKeyword
08 select from Question where QuestionId;
09 end_loop;
10 end_loop;
Transaction Logic for the Creation of a New Ticket

01 ! Access the employee table to verify existence of an employee
02 select from Employee where EmployeeId;
03 if #ProbValidEmployee
04 then ! create ticket record
05 update Ticket num_rows= 1;
06 ! create a record in the TicketEmployee table
07 update TicketEmployee num_rows= 1;
08 ! create records in the TicketKeyword table
09 update TicketKeyword num_rows= #KeywordsPerTicket;
10 end_if;

Transaction Logic for Viewing the Status of Open Tickets

01 ! Get all ticket ids for the employee
02 select from TicketEmployee where EmployeeId;
03 ! Retrieve all open tickets for the employee
04 loop #TicketsPerEmployee
05 select from Ticket where TicketId, Status
06 end_loop;
Estimating Number of I/Os

- Estimate the number of I/Os per database access statement.
  - Consider the existence and types of indexes on the tables.
  - Estimate the number of index blocks accessed.
  - Estimate the number of data pages accessed.

B*-Tree Indexes

key
pointer
index nodes

leaf nodes
B-Tree Calculations

\[ h = \left\lfloor \log_k L \right\rfloor \]

- Height of the B-tree
- Number of entries in the B-tree
- Fanout of the B-tree
- Fill factor: % of useful space of an index page used to store keys and row pointers.

\[ k = \frac{(\text{PageSize} - \text{HeaderSize}) \times f_i}{k_s + \text{rp}} \]

- Size (in bytes) of a key in the index
- Size (in bytes) of a row pointer

Indexes for the Database Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>IndexKey</th>
<th>KeySize(bytes)</th>
<th>Fanout</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question</td>
<td>QuestionId</td>
<td>10</td>
<td>102</td>
<td>2</td>
</tr>
<tr>
<td>Keyword</td>
<td>KeywordId</td>
<td>10</td>
<td>102</td>
<td>2</td>
</tr>
<tr>
<td>Employee</td>
<td>EmployeeId</td>
<td>10</td>
<td>102</td>
<td>3</td>
</tr>
<tr>
<td>Ticket</td>
<td>TicketId</td>
<td>10</td>
<td>102</td>
<td>4</td>
</tr>
<tr>
<td>KeywordQuestion</td>
<td>(KeywordId,QuestionID)</td>
<td>20</td>
<td>59</td>
<td>3</td>
</tr>
<tr>
<td>KeywordQuestion</td>
<td>KeywordId</td>
<td>10</td>
<td>102</td>
<td>3</td>
</tr>
<tr>
<td>TicketEmployee</td>
<td>(TicketId,EmployeeId)</td>
<td>20</td>
<td>59</td>
<td>4</td>
</tr>
<tr>
<td>TicketEmployee</td>
<td>EmployeeId</td>
<td>10</td>
<td>102</td>
<td>4</td>
</tr>
<tr>
<td>TicketKeyword</td>
<td>(TicketId,KeywordId)</td>
<td>20</td>
<td>59</td>
<td>4</td>
</tr>
</tbody>
</table>
Estimating No. I/Os Due to a Select Statement

- Case 1: No index is available. Do a table scan (all data pages have to be accessed).

\[ ndp_T = \left\lfloor \frac{\text{NumRows}_T}{nrp_T} \right\rfloor \]

No. rows of T that can be stored in a data page.

- Case 2: An index is available. The number of I/Os is the height of the B-tree minus 1 (the root is assumed to be in memory) + number of data pages retrieved.

Database Table Sizes

<table>
<thead>
<tr>
<th>Database Table</th>
<th>RowSize(bytes)</th>
<th>Number Rows Per Page</th>
<th>Number Data Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question</td>
<td>1000</td>
<td>1</td>
<td>10,000</td>
</tr>
<tr>
<td>Keyword</td>
<td>50</td>
<td>31</td>
<td>17</td>
</tr>
<tr>
<td>Employee</td>
<td>140</td>
<td>11</td>
<td>1.964</td>
</tr>
<tr>
<td>Ticket</td>
<td>260</td>
<td>6</td>
<td>289,080</td>
</tr>
<tr>
<td>KeywordQuestion</td>
<td>20</td>
<td>73</td>
<td>1,370</td>
</tr>
<tr>
<td>TicketEmployee</td>
<td>20</td>
<td>73</td>
<td>23,760</td>
</tr>
<tr>
<td>TicketKeyword</td>
<td>20</td>
<td>73</td>
<td>118,800</td>
</tr>
</tbody>
</table>

© 2004 D. A. Menascé. All Rights Reserved.
### Computing Number of I/Os

**KeywordsPerQuery**: 2  
**QuestionsPerKeyword**: 20  
**ProbValidEmployee**: 0.9  
**KeywordsPerTicket**: 5  
**TicketsPerEmployee**: 80.3

#### Transaction: FAQ

<table>
<thead>
<tr>
<th>Statement no.</th>
<th>Statement</th>
<th>Type</th>
<th>Table</th>
<th>Where</th>
<th>Index</th>
<th>No. of exec</th>
<th>Prob. Exec</th>
<th>Index I/Os</th>
<th>Data I/Os</th>
<th>Total I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>select</td>
<td>Keyword</td>
<td>Keyword none</td>
<td>2</td>
<td>1.0</td>
<td>0</td>
<td>17</td>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>select</td>
<td>KeywordQuestion</td>
<td>KeywordId</td>
<td>2</td>
<td>1.0</td>
<td>20</td>
<td>44</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>select</td>
<td>Question</td>
<td>QuestionId</td>
<td>40</td>
<td>1.0</td>
<td>1</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Statement 3: Table scan  
2 x 1.0 x 17 = 34

Statement 5: Index-based retrieval  
2 x 1.0 x ((3-1)+20) = 44

---

#### Transaction: New Ticket

<table>
<thead>
<tr>
<th>Statement no.</th>
<th>Statement</th>
<th>Type</th>
<th>Table</th>
<th>Where</th>
<th>Index</th>
<th>No. of exec</th>
<th>Prob. Exec</th>
<th>Index I/Os</th>
<th>Data I/Os</th>
<th>Total I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>select</td>
<td>Employee</td>
<td>EmployeeID</td>
<td></td>
<td></td>
<td>1</td>
<td>1.0</td>
<td>3</td>
<td>3.0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>update</td>
<td>Ticket</td>
<td>TicketId</td>
<td></td>
<td></td>
<td>1</td>
<td>0.9</td>
<td>3</td>
<td>3.0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>update</td>
<td>TicketEmployee</td>
<td>TicketId,EmployeeId</td>
<td></td>
<td></td>
<td>1</td>
<td>0.9</td>
<td>3</td>
<td>3.0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>update</td>
<td>TicketKeyword</td>
<td>TicketId,KeywordId</td>
<td></td>
<td></td>
<td>1</td>
<td>0.9</td>
<td>15</td>
<td>18.0</td>
<td></td>
</tr>
</tbody>
</table>

Statement 2: index-based retrieval  
1 x 1.0 x ((3-1)+1) = 3

---

© 2004 D. A. Menascé. All Rights Reserved.
## Computing Number of I/Os

### Keywords Per Query
- 2

### Questions Per Keyword
- 20

### Probability Valid Employee
- 0.9

### Keywords Per Ticket
- 5

### Tickets Per Employee
- 80.3

### Transaction: View Tickets

<table>
<thead>
<tr>
<th>Statement no.</th>
<th>Statement type</th>
<th>Table</th>
<th>Where</th>
<th>Index</th>
<th>No. of exec</th>
<th>Prob. Exec</th>
<th>Index I/Os</th>
<th>Data I/Os</th>
<th>Total I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>select</td>
<td>TicketEmployee</td>
<td>EmployeeId</td>
<td></td>
<td>1.0</td>
<td>3</td>
<td>80.3</td>
<td>83.3</td>
<td>404.5</td>
</tr>
<tr>
<td>5</td>
<td>select</td>
<td>Ticket</td>
<td>Tickets, Status</td>
<td>80.3</td>
<td>1.0</td>
<td>3</td>
<td>1.0</td>
<td>321.2</td>
<td>404.5</td>
</tr>
</tbody>
</table>

Statement 5: index-based retrieval

\[
80.3 \times 1.0 \times ((4-1)+1) = 321.2
\]

A very large number of I/Os!

## Computing Service Demands

### Service Demand Computations:
- CPU time per 2Kbyte page: 0.0015 sec/IO
- Service time per I/O (sec): 0.008 sec/IO

<table>
<thead>
<tr>
<th>FAQ</th>
<th>Create Ticket</th>
<th>View Tickets</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>0.237</td>
<td>0.046</td>
</tr>
<tr>
<td>Disk</td>
<td>1.264</td>
<td>0.2472</td>
</tr>
</tbody>
</table>

Very large service demand. Disk utilization exceeds 100%.

Solution: instead of storing open and closed tickets in the same table, keep only open tickets in the Tickets table and create another table to archive closed tickets for one year.
The Software Development Life Cycle and Inputs to SPE

- service levels (response times, throughputs, etc)
- hardware/software base (client and software platforms, networking technologies, DBMSs)

- mapping of software modules to C/S architecture
- database design
- networking topology

- number of I/Os and DBMS calls per transaction
- estimates of CPU demand
- estimates of network traffic

- refined I/O and CPU demand estimates
- refined network traffic demand estimates

- measured I/O, CPU, and network service demand estimates

Traditional Software Development Life Cycle

- fix it

- is the performance OK?
- NO

- System Testing
Traditional Software Development Life Cycle

- Common approach:
  - consider Functional Requirements only during development and check Performance Requirements at the end.
  - fix the system if performance is not good!
- Problem:
  - it is very costly and time consuming to fix the problem after the system is ready!
  - fixing the problem may imply in major software rewrites.

Integrating Software Performance Engineering Into the Software Development Life Cycle
Service Demand Generation

\[ D_{i,r} = \sum_{s \in S_{i,r}} n_s \times p_s \times D_{i,r}^s \]

where,

- \( D_{i,r}^s \): average service demand at device \( i \) for class \( r \) due to statement \( s \).
- \( S_{i,r} \): set of statements that generate demands at device \( i \) for class \( r \).
- \( n_s \): average number of time that statement \( s \) is executed.
- \( p_s \): probability that statement \( s \) is executed.