A Utility-Centered Approach to Designing Dependable Internet Services

George Candea, Armando Fox

and other ROC-ers

Stanford University
Motivation

- Tradable properties ("ilities") in system design: functionality, usability, maintainability, performance, portability, security, availability, development cost, ...

- Examples of multiway tradeoffs:
  - Inktomi: data quality $\leftrightarrow$ performance+availability+cost
  - Akamai: security+manageability $\leftrightarrow$ performance+availability+cost
  - Yahoo: cost+portability $\leftrightarrow$ performance+functionality

- Key observation: tradeoffs improve service by providing a better match between service properties and app requirements

- Small systems: right mix is a matter of optimization
  Giant scale: indispensable to the very possibility of building sys
**Issues**

- Making the right tradeoffs is mostly *art*

- 75% of system deployments fail or don’t meet requirements (Yankee Group, 1998)

- Deployment costs exceed expectations (Forrester Research: 25% of Fortune 1000 reported 10-49% higher costs)

- To make it *engineering*, we need three things:
  1. A straightforward model for the design space
  2. Simple, but comprehensive vocabulary for describing properties and the outcome of making tradeoffs
  3. Step-by-step process for trading properties among each other to maximize usefulness of system
Proposed Process

1. **Identify set of relevant axes that span design space in req spec**
   ("spanning set" → any interesting tradeoff can be expressed in terms of the axes)

2. **State system utility functions w.r.t. each axis**

3. **Identify major design areas; choose representative design for each; then**
   - find their coordinates in design space
   - compute overall utility by combining individual utilities

4. **Choose design area that maximizes utility; repeat w/in scope of chosen area**

→ iterative process, with successive refining
Bank of America (http://www.bofa.com)

- System model: service takes inputs and must return outputs within specified amount of time

- Spanning set for design space:
  - Quality of data: consistency with real account
  - Availability: % of requests that are completed as required
  - Performance: Throughput and latency for reads/writes
  - Security: ITSEC levels
  - Cost of ownership: $ amount/year (including initial cost, amortized over expected lifetime of system)
bofa.com: Quality of Data (Fidelity)

- Utility = how useful is a given level of quality
bofa.com: Availability

- Can choose salient points, then interpolate

Utility [normalized]

0 1 2

Availability [%]

0 98 100
bofa.com: Performance/Latency

Utility [normalized]

Max. Latency [sec]

0 10^{-1} 5 10
bofa.com: Performance/Throughput

Utility [normalized]

Min. Throughput [# responses / sec]

0 1000 2000

0 1
bofa.com: Security

Utility [normalized]

0 1 2 3 4 5 6 7

Security [ITSEC EAL]
bdfa.com: Cost

Utility [normalized]

TCO [M$/year]

0 1 3 4
Proposed Process Overview

1. Identify set of relevant axes that span design space ("spanning set" → any interesting tradeoff can be expressed in terms of the axes)

2. State system utility functions with respect to each axis

3. Identify major design areas; choose representative design for each; then
   - find their coordinates in design space
   - compute overall utility by combining individual utilities

4. Choose design area that maximizes utility; repeat w/in scope of chosen area

→ iterative process, with successive refining
**Design Space Navigation: Phase 1**

- Region #1: distributed DB, geographically distributed app servers, distributed web servers, caches everywhere
- Region #2: centralized DB, app server, web servers; no web caches

<table>
<thead>
<tr>
<th>Type</th>
<th>Quality</th>
<th>Availability</th>
<th>Performance Latency</th>
<th>Performance Throughput</th>
<th>Security</th>
<th>Total Cost of Ownership</th>
<th>Overall multiply</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>1.0</td>
<td>0.9 - 1.0</td>
<td>0.9 - 1.0</td>
<td>0.9 – 1.0</td>
<td>0</td>
<td>0.5-0.7</td>
<td>0</td>
</tr>
<tr>
<td>#2</td>
<td>1.0</td>
<td>0.2 - 0.4</td>
<td>0.8 - 0.9</td>
<td>0.6 - 0.8</td>
<td>0 - 1.0</td>
<td>0.7-0.9</td>
<td>0 - 0.26</td>
</tr>
</tbody>
</table>

→ choose Area #2
Design Space Navigation: Phase 2

- Design #1 (w/in Region #1): Sun Solaris 8, Oracle 8i, BEA WebLogic 7.0, Netscape-Enterprise 3.6
- Design #2 (w/in Region #2): RedHat Linux 7.2, proprietary DBMS, proprietary app server, Apache 2.0

<table>
<thead>
<tr>
<th>Type</th>
<th>Quality</th>
<th>Availability</th>
<th>Performance Latency</th>
<th>Performance Throughput</th>
<th>Security</th>
<th>Total Cost of Ownership</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>1.0</td>
<td>0.2 – 0.4</td>
<td>0.8</td>
<td>0.8</td>
<td>0.5 – 1.0</td>
<td>0.7 - 0.8</td>
<td>0.05 – 0.21</td>
</tr>
<tr>
<td>#2</td>
<td>1.0</td>
<td>0.3 - 0.4</td>
<td>0.9</td>
<td>0.8</td>
<td>0 – 0.5</td>
<td>0.8 - 0.9</td>
<td>0 – 0.13</td>
</tr>
</tbody>
</table>

→ choose #1  (much further refinement possible, config, etc.)
Proposed Process Overview

1. Identify set of relevant axes that span design space
   ("spanning set" → any interesting tradeoff can be expressed in terms of the axes)

2. State system utility functions with respect to each axis

3. Identify major design areas;
   choose representative design for each; then
   ● find their coordinates in design space
   ● compute overall utility by combining individual utilities

4. Choose design area that maximizes utility; repeat w/in scope of chosen area

   → iterate until confidence band gets sufficiently narrow
Alternate View

- Design space = multidimensional hyperspace spanned by the axes described earlier and utility as an extra axis
- Candidate designs = “discrete manifold” in this space
- Process of making tradeoffs is analogous to navigating this manifold
- Search for a global max with no cliffs around it (i.e., a smooth plateau) to ensure robustness
- Can break design up into orthogonal subsystems that only concern themselves with subspaces (thus, only some of the axes) makes it easier to design and develop
Benefits: Art vs. Engineering

- Make requirements and tradeoffs more explicit (thus, easier to evaluate and to change later)

- Closer match between requirements and delivered system

- Use for dynamic adaptation (blur design points into regions; at design time you choose region, at runtime you navigate w/in region to choose point)
Difficulties

- Stating utility functions can be a major effort
- Some properties are hard to quantify *(note: we only need to compare them, not measure on some absolute scale)*
- Utility-centered design process may require hierarchical decomposition of axes (typically application-specific) ➔ hierarchical utility composition
- Utility units must be uniform across all axes, to enable comparison
- The comparison must include the ability to say “how much better” one point is than another
- Unlike engineering, where you have struts, bolts, panels, etc. we are far from having standardized components in software engineering
More...

http://RR.stanford.edu