A User-Centric Approach for Improving a Distributed Software System’s Deployment Architecture

Sam Malek

Computer Science Department
University of Southern California
malek@usc.edu
http://www-scf.usc.edu/~malek/
Software Engineering Trends

- Software systems are becoming more complex
- A wider spectrum of distribution and heterogeneity
- Hardware and software mobility is becoming the norm
Software Architecture

- A high-level model of a system
- Represents system organization
  - Components
  - Connectors
  - Events
  - Configurations
Architectural Decisions

♣ Architectural decisions impact non-functional properties of the system

♣ Non-functional property
  ¬ Quality level at which an expected functionality is delivered
  ¬ a.k.a. Quality of Service (QoS)

♣ Making the architectural decisions has remained an art form
  ¬ Lack of quantification and measurement techniques
  ¬ Reliance on domain expert knowledge
Deployment Architecture Impacts QoS

- **Deployment Architecture**: allocation of s/w components to h/w hosts

- h^c deployment architectures are possible for a given system
  - Many provide the same functionality
  - Provide different qualities of service (QoS)
Research Question

- How could we find and effect a deployment architecture that improves (maximizes) multiple QoS dimensions?
  - Where other possible solutions such as caching, hoarding, replication, etc. are not appropriate or ideal

Research Objective

- Devise a solution that is applicable to many classes of application scenarios
  - No particular definition of QoS dimensions, class of systems, etc.
Outline

♣ Problem
  - Motivation
  - Approach
  - Contribution
♣ Deployment analysis
  - Formulation
  - Algorithms
  - Tool support
  - Evaluation
♣ Runtime support
  - Prism-MW
  - Integration
♣ Collaborations and future work
Scenario with a Single QoS Dimension

Objective is to minimize latency

The *optimal* deployment architecture is deployment 1

Most all related approaches stop here
Objective is to minimize latency and maximize durability

There is no optimal deployment architecture!

Phenomenon known as *Pareto Optimal* in multidimensional optimization
Resolving Trade-Offs between QoS Dimensions

♣ A utility function denotes a user’s preferences for a given rate of change in a QoS dimension

♣ Guiding Insight
  - System users have varying QoS preferences for the system services they access

♣ Allows expression of multidimensional optimization in terms of a single scalar value
18 utility functions would have to be considered across 27 deployments.

Challenge: consider many users’ preferences for the many QoS dimensions of many services. “Eyeballing” the solution quickly becomes impossible!
Outline

🌿 Problem
  - Motivation
  - Approach
  - Contributions

🌿 Deployment analysis
  - Formulation
  - Algorithms
  - Tool support
  - Evaluation

🌿 Runtime support
  - Prism-MW
  - Integration

🌿 Collaborations and future work
## Contributions

### Our Approach
- Optimize multiple QoS dimensions and satisfy many constraints
- Consider multiple user-level services
- Generic formal modeling and analysis that can be tailored to each application scenario
- Suite of customizable tools
  - Extension points for configuring the tools
  - Promotes reuse and cross-evaluation of solutions to this problem

### Previous Works
- Availability [Mikic-Rakic 04]
- Remote communication [Bastarica 94, Hunt 99]
- Constraint satisfaction [Menasce 05]
- Assume the entire system provides a single service
- Prescribe a particular model and QoS definition
- Provide algorithms that are not widely applicable
- None that is suitable
  - Do not have the ability to model and relate software, hardware, user, service, and QoS elements
  - Do not provide deployment and monitoring support
Outline

♣ Problem
  ¬ Motivation
  ¬ Approach
  ¬ Contributions

♣ Deployment analysis
  ¬ Formulation
  ¬ Algorithms
  ¬ Tool support
  ¬ Evaluation

♣ Runtime support
  ¬ Prism-MW
  ¬ Integration

♣ Collaborations and future work
 Define sets that specify the system elements and their properties

- Set $C$ of software components
  - $C = \{\text{ResourcesMap, SendMessage, DisplayMap, …}\}$
- Set $CP$ of software component properties
  - $CP = \{\text{size, reliability, …}\}$
- Other sets
  - $H$ of hardware nodes, $N$ of network links, $I$ of logical links, $S$ of services, $Q$ of QoS dimensions, $U$ of users
  - $HP$ of hardware parameters, $NP$ of network link parameters, $CP$ of software component parameters, $IP$ of logical link parameters

 Define functions that quantify system properties

- Function $c\text{Param}:C\_CP\_R$
  - $c\text{Param}(\text{ResourcesMap, size}) = 150$
- Other functions
  - $h\text{Param}:H\_HP\_R$
  - $n\text{Param}:N\_NP\_R$
  - $I\text{Param}:I\_IP\_R$
  - $s\text{Param}:S \_ \{H \cup C \cup N \cup I\} \_ \{HP \cup CP \cup NP \cup IP\} \_ R$
Formal Model of QoS & Users’ Preferences

Define QoS functions

- $qValue: S_Q_DepSpace \_ R$
  - quantifies the achieved level of QoS given a deployment
  - $qValue(Schedule, Latency, Dep 1) = 1ms$

Define users’ preferences in terms of utility

- $qUtil: U_S_Q_R \_ [MinUtil, MaxUtil]$
  - represents the accrued utility for a given rate of change
  - $qUtil(Commander, Schedule, Latency, 0.25) = -1$
A set PC of parameter constraints
- \( PC=\{\text{memory, bandwidth, \ldots}\} \)

A function \( pcSatisfied:PC_{\text{DepSpace}} \rightarrow [0,1] \)
- 1 if constraint is satisfied
- 0 if constraint is not satisfied

Functions that restrict locations of software components
- \( loc:C_{\text{H}} \rightarrow [0,1] \)
  - \( loc(c,h)=1 \) if \( c \) can be deployed on \( h \)
  - \( loc(c,h)=0 \) if \( c \) cannot be deployed on \( h \)
- \( colloc:C_{\text{C}} \rightarrow [-1,1] \)
  - \( colloc(c1,c2)=1 \) if \( c1 \) has to be on the same host as \( c2 \)
  - \( colloc(c1,c2)=-1 \) if \( c1 \) cannot be on the same host as \( c2 \)
  - \( colloc(c1,c2)=0 \) if there are no restrictions
Problem Definition

Given the current deployment \( d \), find an improved deployment \( d' \) such that the users’ overall utility defined as the function

\[
\text{Total utility of changing } d \text{ to } d'
\]

is maximized and specific conditions are satisfied:

1. \( \forall c \in C, \text{loc}(c,H_c)=1 \)

   All location constraints are satisfied

2. \( \forall c_1 \in C, \forall c_2 \in C, \text{if (colloc}(c_1,c_2)=1) \,(H_{c_1}= H_{c_2}), \text{if (colloc}(c_1,c_2)=-1) \,(H_{c_1} \neq H_{c_2}) \)

   All collocation constraints are satisfied

3. \( \forall \text{constr} \in PC \quad \text{pcSatisfied(constr,d)=1} \)

   All system parameter constraints are satisfied
The engineer needs to further refine the “loosely” defined elements of the model

1. Define the pertinent properties of the application scenario
2. Define QoS dimensions in terms of system properties

\[ qValue(s, availability, d) = \sum_{c_1=1}^{C_s} \sum_{c_2=1}^{C_s} sParam(s,I_{c_1,c_2},freq) \times nParam(N_{H_{c_1},H_{c_2}},rel) \]

3. Define system parameter constraints

But how is this done?
- Via the appropriate tool support
Outline

♣ Problem
  - Motivation
  - Approach
  - Contributions

♣ Deployment analysis
  - Formulation
  - Algorithms
  - Tool support
  - Evaluation

♣ Runtime support
  - Prism-MW
  - Integration

♣ Collaborations and future work
A survey of applicable approaches to solving various classes of application scenarios resulted in five types of solutions:

- Two solutions represent state-of-the-art off-the-shelf solutions to solving this problem.
- The remaining solutions are my own algorithmic solutions to address the shortcomings of the off-the-shelf solutions.

Why different algorithms?

- Each algorithm is suitable for a particular class of problems.
Two Off-the-shelf Solvers

1. Mixed Integer Non-linear Programming (MINLP)
   - I represented the problem as a set of linear and non-linear constraint functions
   - Two drawbacks
     • Does not guarantee to find the optimal solution
     • In 20% of large problems, it cannot find any solution

2. Mixed Integer Linear Programming (MIP)
   - Devised an approach to transform our MINLP problem to an MIP problem
   - Developed heuristics to decrease the complexity from $O(2^{|H|^2|C|^2}) \setminus O(|H||C|)$
   - Pros: finds the optimal solution
   - Cons: it is an exponentially complex approach \ infeasible for any sizable system
Three Optimization Algorithms

3. Greedy — Polynomial $O(|S|^3 |C| |U| |Q|)$
   - An iterative algorithm that leverages heuristics for
     - Ranking elements of our problem (e.g., services, hosts, components)
     - Heuristically assigning software components to hardware hosts

4. Genetic — Linear per generation $O(|S| |U| |Q|)$
   - An individual represents a solution to the problem
   - Populations of individuals are evolved via cross-overs and mutations

5. Market-based — Polynomial $O(|C|^2 |S| |U| |Q|)$
   - Autonomous agents on each device auction their local components and bid on each others’ components
   - A good mechanism design (the calculation of auctions and bids) improves the global objective

Common Theme
Heuristically make local decisions that maximize the global objective.
Outline

♣ Problem
  - Motivation
  - Approach
  - Contributions

♣ Deployment analysis
  - Formulation
  - Algorithms
    - Tool support
  - Evaluation

♣ Runtime support
  - Prism-MW
  - Integration

♣ Collaborations and future work
DeSi – Deployment Modeling

- Provides modeling constructs for our problem
- Arbitrary parameters can be associated with these constructs
- A QoS dimension can be defined as a function of the modeling constructs and their parameters
DeSi’s MVC architecture allows for the addition of separate but synchronized views of the underlying model.
New algorithms can be plugged into DeSi to analyze and modify its underlying model.
Outline

♣ Problem
  - Motivation
  - Approach
  - Contributions

♣ Deployment analysis
  - Formulation
  - Algorithms
  - Tool support
  - Evaluation

♣ Runtime support
  - Prism-MW
  - Integration

♣ Collaborations and future work
Algorithms in Practice

<table>
<thead>
<tr>
<th>QoS</th>
<th>MIP</th>
<th>MINLP</th>
<th>Greedy</th>
<th>Genetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>service 1</td>
<td>56%</td>
<td>-8%</td>
<td>18%</td>
<td>-8%</td>
</tr>
<tr>
<td>service 2</td>
<td>93%</td>
<td>94%</td>
<td>97%</td>
<td>24%</td>
</tr>
<tr>
<td>service 3</td>
<td>36%</td>
<td>30%</td>
<td>22%</td>
<td>49%</td>
</tr>
<tr>
<td>service 4</td>
<td>215%</td>
<td>97%</td>
<td>302%</td>
<td>7%</td>
</tr>
<tr>
<td>service 5</td>
<td>65%</td>
<td>7%</td>
<td>26%</td>
<td>26%</td>
</tr>
<tr>
<td>service 6</td>
<td>99%</td>
<td>55%</td>
<td>37%</td>
<td>44%</td>
</tr>
<tr>
<td>service 7</td>
<td>91%</td>
<td>57%</td>
<td>20%</td>
<td>47%</td>
</tr>
<tr>
<td>service 8</td>
<td>43%</td>
<td>22%</td>
<td>7%</td>
<td>56%</td>
</tr>
<tr>
<td>Average</td>
<td>86%</td>
<td>44%</td>
<td>66%</td>
<td>30%</td>
</tr>
<tr>
<td>overallUtil</td>
<td>64</td>
<td>57</td>
<td>55</td>
<td>52</td>
</tr>
</tbody>
</table>

- Results of running the algorithms on an example scenario of 12 Comps, 5 Hosts, 8 Services, and 8 Users
- Significant improvements for all the four QoS dimensions by all the algorithms
- The more important QoS dimensions of services have improved significantly more than others
Algorithms’ Performance

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>MIP</th>
<th>MINLP</th>
<th>Greedy</th>
<th>Genetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>350</td>
<td>5000</td>
<td>17</td>
<td>350</td>
</tr>
<tr>
<td>7</td>
<td>78</td>
<td>350</td>
<td>7</td>
<td>78</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>8</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>20</td>
<td>28</td>
<td>29</td>
<td>20</td>
<td>28</td>
</tr>
<tr>
<td>25C, 10H, 10S, 10U</td>
<td>100000</td>
<td>25600</td>
<td>45</td>
<td>62</td>
</tr>
</tbody>
</table>

Execution Time in Sec. (logarithmic scale)
Algorithms’ Accuracy

![Bar chart showing the objective function values for different problem sizes across various algorithms.

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>MIP</th>
<th>MINLP</th>
<th>Greedy</th>
<th>Genetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>8C, 4H, 4S, 4U</td>
<td>17</td>
<td>14</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>12C, 5H, 5S, 5U</td>
<td>70</td>
<td>64</td>
<td>64</td>
<td>56</td>
</tr>
<tr>
<td>14C, 6H, 6S, 6U</td>
<td>147</td>
<td>122</td>
<td>136</td>
<td>128</td>
</tr>
<tr>
<td>20C, 8H, 8S, 8U</td>
<td>152</td>
<td>157</td>
<td>226</td>
<td>198</td>
</tr>
<tr>
<td>25C, 10H, 10S, 10U</td>
<td>235</td>
<td>226</td>
<td>670</td>
<td>533</td>
</tr>
<tr>
<td>40C, 15H, 15S, 15U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Algorithmic Trade-Offs

♣ Architectural style
  – MIP for constrained styles (e.g., Client-Server), optimization algorithms for flexible styles (e.g., Peer-to-Peer)

♣ Number of QoS dimensions
  – Large number of QoS ◇ Genetic outperforms others

♣ Number of system parameter constraints
  – Large number of constraints ◇ Genetic has a poor accuracy

♣ Stable vs. unstable systems
  – MIP for stable systems, optimization algorithms for unstable systems

♣ Available resources
  – Resource constrained system ◇ Execute the genetic algorithm in parallel on multiple devices

♣ Centralized vs. decentralized systems
  – Decentralized system ◇ Market-based algorithm
Outline

♣ Problem
  ¬ Motivation
  ¬ Approach
  ¬ Contributions

♣ Deployment analysis
  ¬ Formulation
  ¬ Algorithms
  ¬ Tool support
  ¬ Evaluation

♣ Runtime support
  ¬ Prism-MW
  ¬ Integration

♣ Collaborations and future work
Prism-MW

- Prism-MW is an extensible architectural middleware
  - PL-level constructs for architectural concepts

- Extensible design allows for the addition of new facilities

- Developed facilities for
  - (Re)Deployment and (re)configuration in terms of architectural constructs
  - Monitoring both at system and architecture level
Topology Based Routing

♣ Easy redeployment and redistribution of software components onto different hardware configurations
Outline

♣ Problem
  ¬ Motivation
  ¬ Approach
  ¬ Contributions

♣ Deployment analysis
  ¬ Formulation
  ¬ Algorithms
  ¬ Tool support
  ¬ Evaluation

♣ Runtime support
  ¬ Prism-MW
    ¬ Integration

♣ Collaborations and future work
Outline

♣ Problem
  ¬ Motivation
  ¬ Approach
  ¬ Contributions

♣ Deployment analysis
  ¬ Formulation
  ¬ Algorithms
  ¬ Tool support
  ¬ Evaluation

♣ Runtime support
  ¬ Prism-MW
  ¬ Integration

♣ Collaborations and future work
Collaborations with Industrial Partners

♣ US Army
  - Troops Deployment Simulation
  - Large scale search and rescue teams

♣ Boeing
  - Future Combat Systems
  - Determine allocation of software components on virtual hardware platforms

♣ NASA’s Jet Propulsion Lab
  - Dynamic analysis and adaptation of space mission software

♣ Bosch Rsrch. & Tech. Center
  - MIDAS
  - Sensor-network product family
Partial View of MIDAS Architecture
Next Steps

- A comprehensive software architectural trade-off analysis and implementation framework

- What is the impact of other architectural decisions? What is their relationship?

- How should they be modeled, analyzed, and implemented?
Next Steps

♣ Complement static analysis with dynamic analysis

♣ XTEAM
- Supports scenario-driven dynamic analysis
- Provides temporal view of variations in QoS
Selected Publications


6. …
Back Up Slides
A framework that provides
1. an extensible system model
   - inclusion of arbitrary system parameters
   - definition of QoS dimensions using the parameters
   - specification of users’ QoS preferences
2. multiple QoS improvement algorithms
   - different algorithms suited to different classes of systems
3. extensible tool support
   - deployment, execution, and runtime redeployment
   - parameter monitoring and visualization
1. **MINLP – polynomial (N/A)**
   - Represented the problem as a set of (non-)linear constraint functions
   - Does not guarantee the optimal solution or convergence

2. **MIP – exponential: $O(2^{|H|^2|C|^2})$**
   - Devised an approach to transform our MINLP problem to MIP
   - Developed heuristics to decrease complexity to $O(|H|^{|C|})$

3. **Greedy – polynomial: $O(|S|^3 (|C| |U| |Q|)^2)$**
   - An iterative algorithm that leverages several heuristics for
     - Ranking elements of our problem (services, hosts, components, …)
     - Assigning software components to hardware hosts
   - Makes local decisions that often maximize the global objective

4. **Genetic – linear: $O(\#populations \_ \#evolutions \_ \#individuals \_ |S| |U| |Q|)$**
   - An individual represents a solution composed of a sequence of genes
   - A population contains a pool of individuals which are evolved via cross-overs and mutations
   - The accuracy on the representation depends on the ability to promote “good” genes
     - Bad representation does not promote “good” genes \ random search

5. **Market-Based – polynomial: $O(|C|^2 |S| |U| |Q|)$**
   - Software components are auctioned among autonomous agents
   - Auctions are arbitrated in favor of the agents that can maximize the system’s overall utility
Genetic Algorithm - Overview

- An individual represents a solution to the problem
- Each individual is composed of a sequence of genes that represent the structure of the solution
- A population contains a pool of individuals
- An individual for the next generation of the population is evolved in three steps
  - Two or more parent individuals are heuristically selected
  - Via a cross-over between the parent individuals a new individual is created
  - The new individual is mutated via slight random modification of its genes
The problem with this representation is that the genetic properties of parents are not passed on to future generations as a result of cross-overs.

The components that constitute a service are dispersed in the gene sequence of an individual and a cross-over may result in a completely new deployment.
Components of each service are grouped together

If the component participates in more than one service, it is grouped with the most important service

Cross-overs are allowed only on the borders of services

Good solutions are mutated slightly and good genetic properties are promoted
The quality of a population in each evolutionary iteration is improved by selecting parent individuals with a probability that is directly proportional to their fitness values.

Fitness function:
- Returns zero if the individual does not satisfy all of the parameter and locational constraints.
- Otherwise, returns overallUtil for the deployment that corresponds to the individual.

The individual with the highest fitness value is copied directly to the next population:
- Keeping the best individual found in the entire evolutionary search.

The complexity of this algorithm in the worst case is:
Impact of QoS Dimensions

<table>
<thead>
<tr>
<th>Number of QoS dimensions</th>
<th>MIP</th>
<th>MINLP</th>
<th>Greedy</th>
<th>Genetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>130</td>
<td>20</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>192</td>
<td>41</td>
<td>7</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>250</td>
<td>81</td>
<td>11</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>400</td>
<td>132</td>
<td>20</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>602</td>
<td>226</td>
<td>37</td>
<td>27</td>
</tr>
<tr>
<td>6</td>
<td>1017</td>
<td>410</td>
<td>49</td>
<td>31</td>
</tr>
</tbody>
</table>

Execution Time in Seconds (logarithmic scale)
Impact of Heuristics

Mapping in Genetic

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Objective Function Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Genetic without mapping</td>
</tr>
<tr>
<td>10C, 4H, 4S, 4U, 4Q</td>
<td>13</td>
</tr>
<tr>
<td>20C, 8H, 6S, 6U, 4Q</td>
<td>57</td>
</tr>
<tr>
<td>30C, 8H, 8S, 8U, 4Q</td>
<td>73</td>
</tr>
<tr>
<td>40C, 12H, 12S, 12U, 4Q</td>
<td>65</td>
</tr>
<tr>
<td>50C, 15H, 15S, 15U, 4Q</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>Genetic with mapping</td>
</tr>
<tr>
<td>10C, 4H, 4S, 4U, 4Q</td>
<td>48</td>
</tr>
<tr>
<td>20C, 8H, 6S, 6U, 4Q</td>
<td>150</td>
</tr>
<tr>
<td>30C, 8H, 8S, 8U, 4Q</td>
<td>130</td>
</tr>
<tr>
<td>40C, 12H, 12S, 12U, 4Q</td>
<td>192</td>
</tr>
<tr>
<td>50C, 15H, 15S, 15U, 4Q</td>
<td>285</td>
</tr>
<tr>
<td></td>
<td>Genetic with mapping and three parallel executing populations</td>
</tr>
<tr>
<td>10C, 4H, 4S, 4U, 4Q</td>
<td>51</td>
</tr>
<tr>
<td>20C, 8H, 6S, 6U, 4Q</td>
<td>156</td>
</tr>
<tr>
<td>30C, 8H, 8S, 8U, 4Q</td>
<td>132</td>
</tr>
<tr>
<td>40C, 12H, 12S, 12U, 4Q</td>
<td>210</td>
</tr>
<tr>
<td>50C, 15H, 15S, 15U, 4Q</td>
<td>294</td>
</tr>
</tbody>
</table>
Architectural models provide *high-level* concepts
- Components, hosts, links, configurations, etc.

Software systems are implemented using *low-level* PL constructs
- Variables, pointers, procedures, objects, etc.

Deployment is ad-hoc
- Shell scripts, make files, system commands, etc.

Bridging the models to runtime (re)deployment is challenging
- Middleware can help

Existing middleware technologies
- Do not support some architectural concepts (e.g., explicit connectors, configuration)
- Lack support for remote (re)deployment and monitoring facilities

What is needed is “architectural middleware”
class DemoArch {
    static public void main(String argv[]) {
        Architecture arch = new Architecture ("DEMO");
        // create components
        Component a = new CompA ("A");
        Component b = new CompB ("B");
        Component c = new CompC ("C");
        // create connectors
        Connector d = new Connector ("D");
        // add components and connectors
        arch.addComponent(a);
        arch.addComponent(b);
        arch.addComponent(c);
        arch.addConnector(d);
        // establish the interconnections
        arch.weld(a, d);
        arch.weld(b, d);
        arch.weld(d, c);
    }
}
Component C sends an event

Event e = new Event("Event_C", REQUEST);
   e.addParameter("param_1", p1);
   send(e);

Component B handles the event and sends a response

public void handle(Event e) {
   if (e.equals("Event_C")) {
      ... 
      Event e1 = new Event("RSP_to_C", REPLY);
      e1.addParameter("response", resp);
      send(e1);
   }
}...
Event Dispatching

- Topology based routing
- Easy redeployment and redistribution of components onto different hardware configurations