Utility-based Optimal Service Selection for Business Processes in Service Oriented Architectures

Vinod K. Dubey           Daniel A. Menascé

Web Services (ICWS), 2010 IEEE International Conference on. IEEE, 2010

Summarized by: Noor Bajunaid
Introduction

• Service Oriented Architectures allow service providers to provide similar functionalities with different QoS and cost.

• There is a need for server provider selection algorithms that optimize a utility function under constraints, efficiently.
Problem definition

• A business process B, with N activities, $a_i, \ldots, a_n$

  • Maximize $U(E[R(z)], A(z), X(z))$

• subject to

  • $E[R(z)] \leq R_{\text{max}}$
  
  • $A_{\text{min}} \leq A(z) \leq 1$
  
  • $X(z) \geq X_{\text{min}}$
  
  • $C(z) \leq C_{\text{max}}$
  
  • $z \in Z$
<sequence>
  <invoke a1>
  <switch>
    <case q1>
      <flow>
        <invoke a2>
        <sequence>
          <invoke a3>
          <invoke a4>
        </sequence>
      </flow>
    </case q1>
    <case q2=(1-q1)>
      <invoke a5>
    </case q2>
  </switch>
  <invoke a6>
</sequence>
Utility Functions

\[ U_i(v(z)) = K_i \frac{e^{\alpha_i(\beta_i-v(z))}}{1 + e^{\alpha_i(\beta_i-v(z))}} \]

\[ K_i = 1 + \frac{e^{\alpha_i \beta_i}}{e^{\alpha_i \beta_i}} \text{ for execution time} \]

\[ 1 \text{ for throughput} \]

\[ 1 + \frac{e^{\alpha_i (\beta_i - 1)}}{1 + e^{\alpha_i (\beta_i - 1)}} \text{ for availability} \]

\[ U_g(z) = \left( \prod_{i=1}^{3} (U_i(z))^w_i \right) \sum_{j=1}^{3} w_j \]
Computation of End-to-End QoS Metrics

Algorithm 1 Availability Computation of a BPEL process

1: function $A$(node $i$)
2: if label($i$) = leaf node then
3: return $A_i$;
4: else
5: if label($i$) = sequence then
6: return $\prod_{k \in \text{children}(i)} A(k)$;
7: else if label($i$) = switch then
8: return $\sum_{k \in \text{children}(i)} q_k \times A(k)$;
9: else if label($i$) = flow then
10: return $\prod_{k \in \text{children}(i)} A(k)$;
11: end if
12: end if
Availability

\[
A = A_1 \times \\
\{q_1 \times A_2 \times [A_3 \times A_4] \times \\
q_2 \times A_5\} \times \\
A_6
\]
Computation of End-to-End QoS Metrics

Algorithm 2 Throughput Computation of a BPEL process

1: function \( X(\text{node } i) \)
2: if label(i) = leaf node then
3: \hspace{1em} return \( X_i \);
4: else
5: \hspace{1em} if label(i) = sequence then
6: \hspace{2em} return \( \min_{k \in \text{children}(i)} X(k) \);
7: \hspace{1em} else if label(i) = switch then
8: \hspace{2em} return \( \sum_{k \in \text{children}(i)} q_k \times X(k) \);
9: \hspace{1em} else if label(i) = flow then
10: \hspace{2em} return \( \min_{k \in \text{children}(i)} X(k) \);
11: \hspace{1em} end if
12: end if
Throughput

\[ X = \min\{X_1, \ (q_1 \times \min\{X_2, X_3, X_4\}), \ q_2 \times X_5, \ X_6\} \]
Computation of End-to-End QoS Metrics

**Algorithm 1** Compute the execution time of a BPEL process

1: function Compute $R$(node $i$) \\
2: if $i$ is a leaf node then \\
3: return $R_i$; \\
4: else \\
5: for all $k \in$ children($i$) do \\
6: $R_k$ = Compute $R(k)$; \\
7: end for \\
8: if label($i$) = sequence then \\
9: return $R_i = \sum_{k \in \text{children}(i)} R_k$; \\
10: else if label($i$) = switch then \\
11: return $R_i = \sum_{k \in \text{children}(i)} q_i \times R_k$; \\
12: else if label($i$) = flow then \\
13: return $R_i = \max_{k \in \text{children}(i)} R_k$; \\
14: end if \\
15: end if

Execution Time

\[ R = R_1 + q_1 \times \max \{R_2, (R_3 + R_4)\} + q_2 \times R_5 + R_6 \]
Computation of End-to-End QoS Metrics

Algorithm 2 Compute the cost of executing a BPEL process

1: function Compute $C(node \ i)$
2: if $i$ is a leaf node then
3: \hspace{5mm} return $C_i$
4: else
5: \hspace{5mm} for all $k \in$ children($i$) do
6: \hspace{10mm} $C_k =$Compute $C(k)$
7: \hspace{5mm} end for
8: if (label($i$) = sequence) or (label($i$) = flow) then
9: \hspace{5mm} return $C_i = \sum_{k \in \text{children}(i)} C_k$
10: else if label($i$) = switch then
11: \hspace{5mm} return $C_i = \sum_{k \in \text{children}(i)} q_i \times C_k$
12: end if
13: end if

Cost

\[ C = C_1 + q_1 \cdot (C_2 + C_3 + C_4) + q_2 \cdot C_5 + C_6 \]
Optimal Service Selection

1) Extended JOSeS Algorithm:
   • optimal solution
   • efficient for moderate complicity

2) HCB Heuristic Algorithm
   • near-optimal solution
   • efficient even for large set of services
Extended JOSeS Algorithm

- Extends Jensen-based Optimal Service Selection

- Jensen’s inequality:
  \[ E[\max \{R_1, \ldots, R_n\}] \geq \max \{E[R_1], \ldots, E[R_n]\} \]

- It is expensive to compute \( E[\max \{R_1, \ldots, R_n\}] \).

- Jensens’s inequality provides a lower bound that is easier to compute.

- If the lower bound exceeds the maximum execution time, we ignore the allocation and avoid the expensive computation.
Extended JOSeS Algorithm

• If sub-allocation \((s_1, \ldots, s_k)\), \(k \leq N\), violates a constraint, it can be discarded without the need for selecting SPs for activities of order \(> k\).
Extended JOSeS Algorithm

Let \( l_k \) be the list of SPs for \( a_k \): **

- \textbf{next}(k)\footnote{Menascé, Daniel A., Emiliano Casalicchio, and Vinod Dubey. "On optimal service selection in service oriented architectures." \textit{Performance Evaluation} 67.8 (2010): 659-675.} returns the next, not yet evaluated, SP in \( l_k \), or returns null if all the SPs in \( l_k \) were already evaluated.

- \textbf{reset}(k)\footnote{Menascé, Daniel A., Emiliano Casalicchio, and Vinod Dubey. "On optimal service selection in service oriented architectures." \textit{Performance Evaluation} 67.8 (2010): 659-675.} sets all SPs in all lists \( l_j \) (\( j = k, \ldots, N \)) as “not-visited”.
Extended JOSeS Algorithm

S\textsubscript{1,1} S\textsubscript{1,2} S\textsubscript{1,3}
S\textsubscript{2,1} S\textsubscript{2,2}
S\textsubscript{3,1} S\textsubscript{3,2} S\textsubscript{3,3}

S\textsubscript{1,1} S\textsubscript{1,2} S\textsubscript{1,3}
S\textsubscript{2,1} S\textsubscript{2,2}
S\textsubscript{3,1} S\textsubscript{3,2} S\textsubscript{3,3}

S\textsubscript{1,1} S\textsubscript{1,2} S\textsubscript{1,3}
S\textsubscript{2,1} S\textsubscript{2,2}
S\textsubscript{3,1} S\textsubscript{3,2} S\textsubscript{3,3}

S\textsubscript{1,1}
S\textsubscript{1,1} S\textsubscript{2,1} violation
Extended JOSeS Algorithm

S_{1,1} S_{1,2} S_{1,3}

S_{2,1} S_{2,2}

S_{3,1} S_{3,2} S_{3,3}

S_{1,1} S_{1,2} S_{1,3}

S_{2,1} S_{2,2}

S_{3,1} S_{3,2} S_{3,3}

S_{1,1} S_{1,2} S_{1,3}

S_{2,1} S_{2,2}

S_{3,1} S_{3,2} S_{3,3}

S_{1,1} S_{2,2}

S_{1,1} S_{2,2}

S_{3,1}
Extended JOSeS Algorithm

Allocations that violate constraint will reduce the number of examined points
HCB Heuristic Algorithm

• Hill-climbing based:
  • Define a neighborhood of an allocation
  • Move to the best allocation in the neighborhood
  • Repeat until near-optimum solution is found or maximum number of starts
HCB Heuristic Algorithm

• Neighborhood:

  • for each activity, replace the SP with the other SPs that will maximize improvement in each QoS metric

\[ s^R = \arg \max_{k=1}^{\mid S_i \mid} \{ 1 - \frac{R_{i,k}}{R_{i,curr}} \} \]

\[ s^A = \arg \max_{k=1}^{\mid S_i \mid} \{ \frac{A_{i,k}}{A_{i,curr}} - 1 \} \]

\[ s^X = \arg \max_{k=1}^{\mid S_i \mid} \{ \frac{X_{i,k}}{X_{i,curr}} - 1 \} \]
HCB Heuristic Algorithm

Algorithm 6 Identify Neighbors

1: function neighbors (z₀) returns (Z)
2:  Z ← ∅; /* Initialize with empty neighborhood */
3:  N ← ∅; /* All neighbors */
4:  for all activity i = 1,...,N do
5:      for all qᵢ ∈ {Rᵢ, Aᵢ, Xᵢ} do
6:          if qᵢ = Rᵢ then
7:              /* s = best improvement in response time */
8:              s = arg maxⱼ∈[Sᵢ] {1 − qᵢⱼ / qᵢ,cᵢⱼ};
9:          else
10:             /* s = best improvement in availability and throughput */
11:             s = arg maxⱼ∈[Sᵢ] \{\frac{qᵢⱼ}{qᵢ,cᵢⱼ} − 1\};
12:         end if
13:         z = replace (z₀, i, s); /* Replace current SP of aᵢ in z₀ by s */
14:         if z ∉ N then
15:             N ← N ∪ z;
16:             if (((C(z) ≤ C_{max}) and (A(z) ≥ A_{min}) and
17:                 (E[R(z)] ≤ R_{max}) and (X(z) ≥ X_{min}))
18:                     then
19:                         if (E[R(z)] ≤ R_{max}) then
20:                             Z ← Z ∪ z;
21:                         end if
22:                     end if
23:                 end if
24:             end if
25:         end for
26:      end for
27:  end for
28:  return Z;


Algorithm 5 HCB Heuristic Algorithm

1: function HeuristicSolution() returns (z)
2: nrestarts ← 0;
3: while (nrestarts < maxrestarts) do
4:   z₀ ← randomStart(); /* random start */
5:   nrestarts ← nrestarts + 1; searching ← TRUE;
6:   while (searching) do
7:     Z ← neighbors (z₀); /* get feasible neighbors */
8:     zopt ← arg max_{z_i ∈ Z} {U(z_i)}; /* Identify neighbor with highest utility */
9:     if (U(zopt) > U(z₀)) then
10:        z₀ ← zopt;
11:     else
12:        searching ← FALSE; /* local optimum */
13:     end if
14:   end while
15:   if (nrestarts = 1) then
16:      zopt ← z₀;
17:   else if (U(zopt) > U(zopt)) then
18:      zopt ← zopt;
19:   end if
20: end while
21: return zopt;
22: end function
Experimental Evaluation

1. Determine how effective is the heuristic solution compared to the optimal.

2. Compare the number of points examined by each algorithm

3. Compare both algorithms over a wide range of parameters
Experimental Evaluation

• 50 BPEL business processes.

• 6 - 9 activities with different construct (sequence, flow, switch)

• 2 - 7 SPs per activity.

• Constraints strength varied from 10% to 40%

• each combination was ran through JeSOS once, and through HCB 30 times
Experimental Evaluation

- QoS metrics of each SP for each activity are given:

$$(E[R], A, X)$$

$C_{Total} = C(r) + C(X) + C(A)$$
Experimental Evaluation

Stricter constraints reduce the size of the neighborhood and decrease the breadth of the search.

Figure 1. Average $U_h/U_o$ (%) vs. nspa for four constraint strengths
Experimental Evaluation

As CS increases, more sub-allocations are prematurely declared unfeasible. JeSOS will examine significantly less points and take less time.

Figure 2. Average number of points examined $N_h$ and $N_o$ vs. $nspa$ for four constraint strengths

Figure 3. Average computation time $T_h$ and $T_o$ vs. $nspa$ for four constraint strengths
For a complex Business process and 7 SPs, HCB achieved 99.97% of optimal utility by examining 100 points. JeSOS examined more than 10,000,000 points!
Experimental Evaluation

- HCB scalability for large number of SPs/activity (50 - 400).

- Regression shows that the number of examined points increases linearly with number of SPs.

![Graph showing the relationship between the average number of points examined by heuristic and the number of SPs per activity. The regression line is given by \( N_h = 39 \times N_{SPA} \) with \( R^2 = 0.99678 \).]

Figure 6. Average \( N_h \) vs. \( n_{spa} \)
Conclusion

- The most important conditions for JeSOS to be efficient are:
  - simple business process structure.
  - limited number of server providers
  - stronger constraints.
- HCB is potentially very efficient for autonomic near-optimal resource allocation.