Supplementary Material for the Paper *Probabilistic QoS-aware Placement of VNF chains at the Edge* published in Theory and Practice of Logic Programming

STEFANO FORTI, FEDERICA PAGANELLI, ANTONIO BROGI

Department of Computer Science, University of Pisa
stefano.forti@di.unipi.it, {antonio.brogi, federica.paganelli }@unipi.it

Appendix A

1 EdgeUsher Prototype Implementation

The complete code (72 sloc) of EdgeUsher, as presented in the article, follows.

```
placement(Chain, Placement, ServiceRoutes) :-
    chain(Chain, Services),
    servicePlacement(Services, Placement, []),
    flowPlacement(Placement, ServiceRoutes).

servicePlacement([], [], _).
servicePlacement([S|Ss], [on(S,N)|P], AllocatedHW) :-
    service(S, _, HW_Reqs, Thing_Reqs, Sec_Reqs),
    node(N, HW_Caps, Thing_Caps, Sec_Caps),
    HW_Reqs =< HW_Caps,
    thingReqsOK(Thing_Reqs, Thing_Caps),
    secReqsOK(Sec_Reqs, Sec_Caps),
    hwReqsOK(HW_Reqs, HW_Caps, N, AllocatedHW, NewAllocatedHW),
    servicePlacement(Ss, P, NewAllocatedHW).

thingReqsOK(T_Reqs, T_Caps) :- subset(T_Reqs, T_Caps).

secReqsOK([],_).
secReqsOK([SR|SRs], Sec_Caps) :- subset([SR|SRs], Sec_Caps).
secReqsOK(and(P1,P2), Sec_Caps) :- secReqsOK(P1, Sec_Caps), secReqsOK(P2,
     Sec_Caps).
secReqsOK(or(P1,P2), Sec_Caps) :- secReqsOK(P1, Sec_Caps); secReqsOK(P2,
     Sec_Caps).
secReqsOK(P, Sec_Caps) :- atom(P), member(P, Sec_Caps).

hwReqsOK(HW_Reqs, _, N, [], [(N,HW_Reqs)]).
hwReqsOK(HW_Reqs, HW_Caps, N, [(N,A)|As], [(N,NewA)|As]) :-
    HW_Reqs + A =< HW_Caps, NewA is A + HW_Reqs.
hwReqsOK(HW_Reqs, HW_Caps, N, [(N1,A1)|As], [(N1,A1)|NewAs]) :-
    N \== N1, hwReqsOK(HW_Reqs, HW_Caps, N, As, NewAs).

flowPlacement(Placement, ServiceRoutes) :-
    findall(flow(S1, S2, Br), flow(S1, S2, Br), ServiceFlows),
    flowPlacement(ServiceFlows, Placement, [], ServiceRoutes, [], S2S_latencies
    ),
    maxLatency(LChain, RequiredLatency),   %hp: only one maxLatency def
    latencyOK(LChain, RequiredLatency, S2S_latencies).

flowPlacement([], _, SRs, SRs, Lats, Lats).
flowPlacement([flow(S1, S2, _)|SFs], P, SRs, NewSRs, Lats, NewLats) :-
```

```
    subset([on(S1,N), on(S2,N)], P),
    flowPlacement(SFs, P, SRs, NewSRs, [(S1,S2,0)|Lats], NewLats).
flowPlacement([flow(S1, S2, Br)|SFs], P, SRs, NewSRs, Lats, NewLats) :-
    subset([on(S1,N1), on(S2,N2)], P),
    N1 \== N2,
    path(N1, N2, 2, [], Path, 0, Lat),
    update(Path, Br, S1, S2, SRs, SR2s),
    flowPlacement(SFs, P, SR2s, NewSRs, [(S1,S2,Lat)|Lats], NewLats).

path(N1, N2, Radius, Path, [(N1, N2, Bf)|Path], Lat, NewLat) :-
    Radius > 0,
    link(N1, N2, Lf, Bf),
    NewLat is Lat + Lf.

path(N1, N2, Radius, Path, NewPath, Lat, NewLat) :-
    Radius > 0,
    link(N1, N12, Lf, Bf), N12 \== N2, \+ member((N12,_,_,_), Path),
    NewRadius is Radius-1,
    Lat2 is Lat + Lf,
    path(N12, N2, NewRadius, [(N1, N12, Bf)|Path], NewPath, Lat2, NewLat).

update([],_,_,_,SRs,SRs).
update([(N1, N2, Bf)|Path], Br, S1, S2, SRs, NewSRs) :-
    updateOne((N1, N2, Bf), Br, S1, S2, SRs, SR2s),
    update(Path, Br, S1, S2, SR2s, NewSRs).

updateOne((N1, N2, Bf), Br, S1, S2, [], [(N1, N2, Br, [(S1,S2)])]) :-
    Br =< Bf.
updateOne((N1, N2, Bf), Br, S1, S2, [(N1, N2, Bass, S2Ss)|SR], [(N1, N2, NewBa,
      [(S1,S2)|S2Ss])|SR]) :-
    Br =< Bf-Bass, NewBa is Br+Bass.
updateOne((N1, N2, Bf), Br, S1, S2, [(X, Y, Bass, S2Ss)|SR], [(X, Y, Bass, S2Ss
    )|NewSR]) :-
    (N1 \== X; N2 \== Y),
    updateOne((N1, N2, Bf), Br, S1, S2, SR, NewSR).

latencyOK(LChain, RequiredLatency, S2S_latencies) :-
    chainLatency(LChain, S2S_latencies, 0, ChainLatency),
    ChainLatency =< RequiredLatency.

chainLatency([S], _, Latency, NewLatency) :-
    service(S, S_Service_Time, _, _, _),
    NewLatency is Latency + S_Service_Time.
chainLatency([S1,S2|LChain], S2S_latencies, Latency, NewLatency) :-
    member((S1,S2,Lf), S2S_latencies),
    service(S1, S1_Service_Time, _, _, _),
    Latency2 is Latency+S1_Service_Time+Lf,
    chainLatency([S2|LChain], S2S_latencies, Latency2, NewLatency).
```

2 Proof of Correctness and Termination of EdgeUsher

We include here a sketch of the proofs of termination and correctness of EdgeUsher.

Proposition 1. The query $placement(Chain, Placement, ServiceRoutes)$ always terminates.

Proof. It is easy to prove that the query $placement(Chain, Placement, ServiceRoutes)$ always terminates since it:

- calls *chain*/2, which is matched against a set of facts and terminates immediately,
- calls *servicePlacement*/2 and *flowPlacement*/2, which both terminate.

The call $servicePlacement(Services, Placement)$ terminates since:

- predicate *servicePlacement*/2 just calls *servicePlacement*/3,

- *servicePlacement*/3 performs tail-recursion by reducing the size of its first term (a list), so that if the size of the first term in the first call to *servicePlacement*/3 is n then *servicePlacement*/3 performs n tail-recursive calls and terminates,
- before tail-recurring, *servicePlacement*/3
  - calls *service*/5 and *node*/4, which are both matched against a set of facts and terminate immediately,
  - calls *thingReqsOK*/2, which scans m times its second term (a list), where m is the size of its first term (a list, too), and terminates,
  - calls *secReqsOK*/2, which terminates
    - either after scanning m times its second term (a list), where m is the size of its first term (if it is a list)
    - or after recurring m times by reducing the size of its first term (if it is an and-or term of depth m) and after scanning m times its second term (a list),
  - calls *hwReqsOK*/5, which performs tail-recursion by reducing the size of its fourth term (a list), and terminates.

The call $flowPlacement(Placement, ServiceRoutes)$ terminates since it:

- calls $findall$/3, whose inner goal is matched against a set of facts and terminates,
- calls $flowPlacement$/6, which performs tail-recursion by reducing the size of its first term (a list), and terminates; before tail-recurring, $flowPlacement$/6
  - calls *subset*/2, which scans twice its second term (a list),
  - calls *path*/7, which performs tail-recursion by reducing the size of its third term (a natural number), and terminates,
  - calls *update*/6, which performs tail-recursion by reducing the size of its first term (a list), and terminates,
  - before tail-recurring, *update*/6 calls *updateOne*/6, which performs tail-recursion by reducing the size of its fifth term (a list), and terminates
- calls *maxLatency*/2, which is matched against a set of facts and terminates immediately,
- calls *latencyOK*/3, which just calls *chainLatency*/4
  - *chainLatency*/4 performs tail-recursion by reducing the size of its first term (a list), and terminates
  - before tail-recurring, *chainLatency*/4 calls *service*/5 (which is matched against a set of facts and terminates immediately) and scans once its second term (a list). ◇

Proposition 2. If $servicePlacement([s_1,\dots,s_k],P)$ is proved with computed answer substitution $P = [on(s_1,n_1),\dots,on(s_k,n_h)]$, then the service placement defined by P satisfies all the IoT, security and hardware requirements of $[s_1,\dots,s_k]$.

Proof. We first prove —by induction on the size of the first term of $servicePlacement([s_1,\dots,s_k], P)$ — that:

(*) if $servicePlacement([s_1,\dots,s_k],P) \to^*$
$servicePlacement([],[on(s_1,n_1),\dots,on(s_k,n_h)],[(n_1,hw_1),\dots,(n_h,hw_h)]) \to true$
then $\forall j \in [1,h] : hw_j = \sum_{on(s_i,n_j)} hw\_reqs(s_i) \leq hw\_caps(n_j)$

(Base case) Trivial since if $servicePlacement([s_1],Placement) \to^*$
$servicePlacement([],[on(s_1,n_1)],[(n_1,hw_1)]) \to true$
then $hw_1 = hw\_reqs(s_1) \leq hw\_caps(n_1)$, by lines 13 and 24 of the code in the previous section.

(Inductive case)
If $servicePlacement([s_1,\dots,s_k,s_{k+1}],Placement)$

$\rightarrow^*$ *servicePlacement*$([s_{k+1}],[on(s_1,n_1),\ldots,on(s_k,n_h)],[(n_1,hw_1),\ldots,(n_h,hw_h)])$
$\rightarrow^*$ *servicePlacement*$([],[on(s_1,n_1),\ldots,on(s_k,n_h),on(s_{k+1},n_{h+1})],[(n_1,hw_1),\ldots,(n_h,hw_h),$
$(n_{h+1},hw_{h+1})])$
$\rightarrow$ *true*
where $n_{h+1} \notin \{n_1,\ldots,n_h\}$ then
$\forall j \in [1,h] : hw_j = \sum_{on(s_i,n_j)} hw\_reqs(s_i) \leq hw\_caps(n_j)$ by inductive hypothesis,
and $hw_{h+1} = hw\_reqs(s_{k+1}) \leq hw\_caps(n_{h+1})$ by lines 13 and 24, 27, 28 of the code in the previous section.
If *servicePlacement*$([s_1,\ldots,s_k,s_{k+1}],Placement)$
$\rightarrow^*$ *servicePlacement*$([s_{k+1}],[on(s_1,n_1),\ldots,on(s_k,n_h)],[(n_1,hw_1),\ldots,(n_h,hw_h)])$
$\rightarrow^*$ *servicePlacement*$([],[on(s_1,n_1),\ldots,on(s_k,n_h),on(s_{k+1},\overline{n})],[(n_1,hw'_1),\ldots,(n_h,hw'_h)]) \rightarrow$ *true*
where $\overline{n} \in \{n_1,\ldots,n_h\}$ then
$\forall j \in [1,h]$: $n_j \neq \overline{n} \Rightarrow hw'_j = hw_j = \sum_{on(s_i,n_j)} hw\_reqs(s_i) \leq hw\_caps(n_j)$ by inductive hypothesis, and
$n_j = \overline{n} \Rightarrow hw'_j = \sum_{on(s_i,n_j)} hw\_reqs(s_i) \leq hw\_caps(n_j)$

by inductive hypothesis and by lines 24—26 of the code in the previous section.

We now prove that if *servicePlacement*$([s_1,\ldots,s_k],P) \rightarrow^*$ *true* with computed answer substitution $P = [on(s_1,n_1),\ldots,on(s_k,n_h)]$ then the service placement defined by P satisfies all the IoT, security and hardware requirements of $[s_1,\ldots,s_k]$.
The proof is by induction on the size of the first term of *servicePlacement*$([s_1,\ldots,s_k],P)$.
(Base case) If *servicePlacement*$([s_1],P) \rightarrow^*$ *true* with computed answer substitution $P = [on(s_1, n_1)]$ then P satisfies the IoT, security and hardware requirements of $[s_1]$ by lines 16 and 19—22 of the code in the previous section, and by (*) respectively.
(Inductive case) If *servicePlacement*$([s_1,\ldots,s_k],P) \rightarrow^*$ *true* with computed answer substitution $P = [on(s_1,n_1),\ldots,on(s_k,n_h)]$ then P satisfies all the IoT, security and hardware requirements of $[s_1,\ldots,s_k]$ by inductive hypothesis and by lines 16 and 19—22 of the code in the previous section and by (*), respectively. $\diamond$