A Model and Algorithm for Self-Adaptation in Service-oriented Systems

Christoph Dorn, Daniel Schall, Schahram Dustdar
Distributed Systems Group
Vienna University of Technology, 1040 Vienna, Austria
lastname@infosys.tuwien.ac.at

***Abstract*—In this paper, we address the problem of self-adaptation in internet-scale service-oriented systems. Services need to adapt by select the best neighboring services solely based on local, limited information. In such complex systems, the global significance of the various selection parameters dynamically changes. We introduce a novel metric measuring the distribution and potential impact of service properties affecting such selection parameters. We further present an formalism identifying the most significant properties based on aggregated service interaction data. We ultimately provide a ranking algorithm exploiting these dynamic interaction characteristics. Experimental evaluation demonstrates scalability and adaptiveness of our approach.**

I. Introduction

Internet-scale service-oriented systems contain thousands of services distributed across multiple organizations. These services collaborate to achieve a common goal without relying on centralized control. Such systems are subject to dynamic changes but have become too complex to be managed by human administrators [1]. Services need to become self-adaptive [2] to maintain the system's functionality.

One adaptation technique is dynamically selecting the best service to forward a request to. Due to scale, each service maintains connections only to a neighboring subset of all services. We refer to the complete set of services and their connections as a service network. In such an environment, no service has a complete picture of the overall network. One fundamental problem is identifying the most significant global selection parameters given local information only.

Autonomic computing [3] in general—and several service-oriented computing approaches [4], [5], [6] in particular—address self-adaptive behavior. These exhibit, however, two fundamental shortcomings: (i) they require complete, global information and (ii) they apply a stable set of adaptation parameters. This paper specifically addresses these two aspects.

Our main contribution is a model and algorithm to enable self-adaptation in service-oriented systems. Specifically, we determine the most relevant services to forward an invocation request to. To this end, we identify and analyze potential service properties with the most significant effect on service interactions. We focus on observable, public service properties (e.g., organization, location, type, capabilities) that become selection parameters when exhibiting a measurable effect on service interactions (i.e., accepted/rejected invocations). We define the impact of a property as the extent to which services of one property value (e.g., location A) successfully forward requests to services exhibiting a different property value (e.g., location B).

We structure this paper as follows: Section II gives a motivating example before presenting our approach in detail. Section III discusses related work. Section IV introduces an entropy model to measure service property distribution within a service-oriented system. We subsequently provide an algorithm in Section V for evaluating the impact of specific property values on the service interaction structure. The ranking algorithm in Section VI utilizes the impact magnitude to recommend suitable neighbor services. The evaluation of our approach in Section VII relies on simulation of a service network. Finally, conclusions and future work complete this paper.

II. Towards Self-Adaptation

Two broad design principles aim for self-adaptive behavior. Autonomic systems implementing the MAPE-K cycle (Monitor, Analyze, Plan, Execute, Knowledge) [2] require a global view of the system to enforce optimal adaptation actions [7]. Socially and biology-inspired systems exploit emerging phenomena [8]. The collective behavior of system elements yields global desirable goals purely based on local information.

Our approach combines these two design principles. We apply local service properties and service interaction data captured by distributed logging [9], monitoring [10], or sensing [11] mechanisms. Analysis and planning (of the MAPE-K cycle) apply global—but aggregated—knowledge while the ultimate execution steps (i.e., service ranking) are triggered by individual services. This paper focuses primarily on the formalism for analyzing and planning adaptive service selection. Unfortunately, we cannot discuss detailed engineering aspects due to page restrictions.

A. Motivating Scenario

Assume a data service provider participating in a global service network. An example research center becomes a customer in the early phases of a data-intensive project. At the beginning, the need for extensive storage space is low, retrieval requests origin at a single location, and updates

occur frequently. Thus requests will mostly happen within the provider's own service network, locally concentrated.

The service interaction characteristics change once data intensive research results are made available for a broader audience. Requests cross provider boundaries, access to data occurs from multiple locations, while updates decrease.

Suppose a new storage service is about to join the network. It does not know the clients it will serve. It is also unaware of the particular service interaction characteristics when serving these clients. The new service, however, needs to learn of the most significant properties to optimally select amongst the existing services for storing and querying data in the service network. For the remainder of this paper, we discuss our approach and findings in the scope of one client for sake of clarity.

B. Approach

For a freshly added service, the significant services are the ones most likely to accept forwarded requests. To this end, we need to identify the properties that determine whether a request is accepted or not. Our approach, thus, focuses on public service information and observable service interactions. In the early stages of our scenario, services with versioning capability are suitable receivers. In later stages, services at remote service providers (i.e., different organizations) or different locations provide most benefit by distributing load.

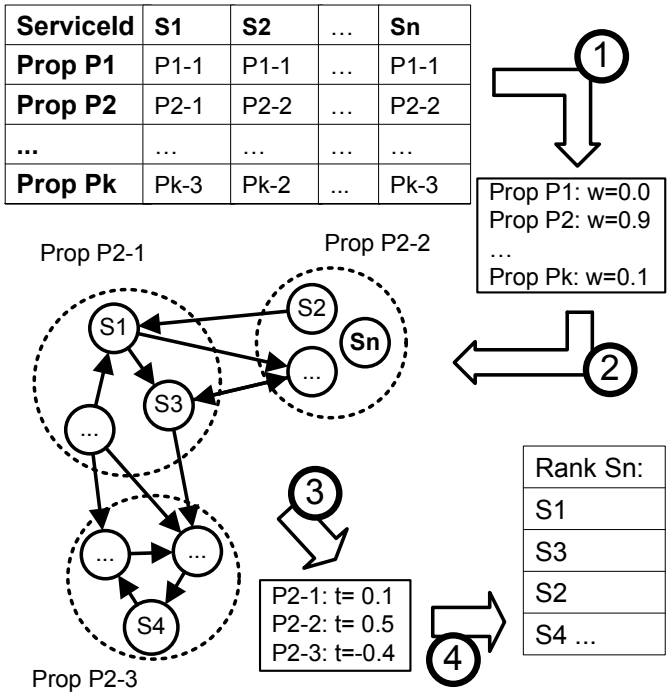


Fig. 1. Property checking, evaluation, and ranking

Figure 1 visualizes the approach comprising the following steps: based on the distribution of property values across services, we derive candidate properties (1). These candidates yield high potential impact on service interaction. For example, when all observed services reside within a single data center, location yields no interaction impact. Similarly, the service identifier property yields no impact neither, as every service exhibits a distinct ID. Thus, neither location nor service identifier become candidates.

Any changes in service properties (including new/leaving services) trigger recalculation of candidate properties. As long as requests traverse only services of one provider, *Organization* will not become a candidate. Once the customer in our example enables access to data for 3rd parties, requests from external services will occur. As multiple organization values emerge, the organization property becomes a potential selection parameter.

The subsequent detailed interaction analysis (2) considers only the properties with highest potential impact (e.g., versioning capability). The analysis focus will shift to other properties when their potential impact measurement significantly changes.

Interaction analysis determines whether services tend to interact with services exhibiting the same or different properties (3). This process runs regularly to maintain up-to-date impact measurements. In the early stages of our scenario, services without versioning capability will forward requests to services with versioning capability. These in turn, will forward only between their kind.

Impact magnitude influences the final ranking order of suitable services (4). The versioning capability will exhibit highest impact on the ranking result, when forwarding occurs only from non-versioning to versioning services. Later in the scenario, we replace the ranking criteria as capabilities become less significant, while spatial and organizational properties emerge.

The motivating scenario involves only a sample set of properties. We provide a generic formalism to determine impact potential and actual impact magnitude for any type of properties. We briefly review related work before providing the underlying mathematical model and algorithms.

III. Related Work

Most frameworks and algorithms rely on a-priori definition of relevant adaptation parameters. Related work in the domain of autonomic computing [3] mostly requires either complete global information or suffices with limited local context.

Quitadamo et al. [4] demonstrate a knowledge-network driven approach to service selection and aggregation. Andreolini et al. [12] exploit load trends for autonomic request forwarding between geographically distributed systems.

Maximilien and Singh [13] combine QoS parameters and trust based on service interactions to find the best service. Focusing purely on self-adjusting trust, they assume fixed individual preferences that do not change under different conditions. This shortcoming is partially addressed by Bottaro and Hall [14] who focus on event-driven adaptive service selection using, however, only local context information.

Colman [15] proposes a hybrid approach to self-organization services through hierarchical structuring of autonomic managers and services. We expect services to self-

organize their interactions, whereas Colman requires the autonomic manager to monitor and control all composed services, thereby severely limiting the size of manageable service compositions.

Jennings et al. [16] discuss an architecture for autonomic management of communication networks. They suggest applying the MAPE-K cycle to a complete set of entities, thus limiting the architecture's applicability to domains exhibiting a central set of goals.

Goal-oriented approaches to dynamic service selection require either a complete global view on all involved entities [5], [6] or utilize local context only [17].

Emergence-based, self-organizing approaches in the domain of autonomic communications [18] inherently lack a clear distinction between regular component functionality and autonomic adaptation.

In Self-Configuring Socio-Technical Systems [19], Bryl and Giorgini describe a multi-agent system reacting to dynamic reconfiguration needs. Saffre et al. [20] present an algorithm that results in self-organizing behavior of services. Membership properties enable the algorithm to achieve the desirable behavior using again only local context information. However, the type and impact of context information is a-priori defined.

Dynamically identifying the most relevant parameters for self-adaptation includes research by Zhang and Figueiredo [21]. Their bayesian network based autonomic feature selection, however, focuses exclusively on service internal measurements and thus neglects any form of interaction data. Marinescu et al. [22] measure the importance of properties on system self-organization, but focus on the impact of gene diversity.

IV. Property Entropy Model

In large-scale networks, service interaction analysis is a computationally intensive task. Knowing which aspects will yield the most significant findings maximizes the efficiency of the analysis process. The primary purpose of a suitable metric is thus to identify those properties, that potentially have a measurable impact on interactions. Such a metric must work on properties consisting of any number of values, and enable comparison of properties that differ in their amount of values. Example service properties include the organization deploying the service, the service location, storage capacity, and request routing capability (e.g., none, random neighbor, round-robin).

The following model and entropy metric calculates the distribution of properties across services. Table I gives a summarized explanation of the symbols applied in the model and impact algorithms.

The metric output for each property is in the interval $[0, 1]$. A metric value v towards 0 describes a trend of services sharing the same property values, while a metric value towards 1 denotes services exhibiting individual property values. Extreme cases include all services having the same property value ($v = 0$) and each service having a distinct property value ($v = 1$).

Symbol	Meaning
$\mathcal{S}$	the set of services $s \in \mathcal{S}$ in a service network $\mathcal{N}$
$\mathcal{P}$	the set of public properties in the service network $\mathcal{N}$
P	a particular public property $P \in \mathcal{P}$ comprising any number of property values $p_i \rightarrow p_n \in P$
$\mathcal{F}$	a function mapping each service s to one property value p for each public property P
$PDE(S, P)$	the property distribution entropy for particular property P and service set $\mathcal{S}$
$PDE_{lower\|upper}$	a function describing the minimum (maximum) PDE values for a given number of property values $p \in P$
$util_{upper\|lower}$	a function describing the minimum (maximum) utility along the lower (upper) PDE limits.
E	set of interaction edges in the directed service interaction graph $\mathcal{G}$
$cluster_P(i)$	set of services exhibiting the same property value $p_i \in P$
$trend_P(i)$	interaction focus (internal or external) of a cluster associated to $p_i \in P$
$imp_P(i)$	interaction impact of a cluster associated to $p_i \in P$
imp_P	overall interaction impact of property P
d	iteration count within the zero model analysis

TABLE I
Symbols applied in the entropy model (upper section) and evaluation algorithm (lower section).

In our model, a service network $\mathcal{N}(\mathcal{S}, \mathcal{P})$ is defined as a set of services $\mathcal{S}$ exhibiting a set of public properties $\mathcal{P}$. Each property $P \in \mathcal{P}$ consist of a set of non-overlapping property values $p_1 \dots p_n$. In addition, for each property P there exists a mapping $\mathcal{F}(\mathcal{S} \mapsto \mathcal{P})$ such that each service $s \in \mathcal{S}$ is assigned to exactly one value instance $p \in P$, ultimately establishing the property matrix (Figure 1 input to Step 1). Whether properties are numerical values, strings, or enumerations is irrelevant as the function $\mathcal{F}$ establishes a unambiguous mapping. For each property P, we define the Property Distribution Entropy (PDE) as follows:

$$\mathrm{PDE}(S, P) = 1 - \sum_{i=1}^{n} \binom{|p_i|}{2} * \binom{z}{2}^{-1} \tag{1}$$

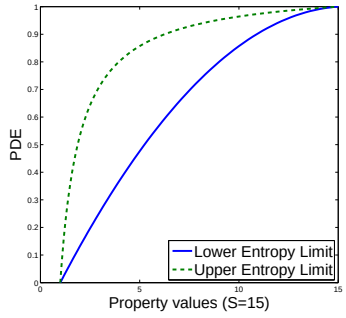
where $|p_i|$ is the number of services mapped to property value $p_i \in P$ and $z = |S|$ is the total number of services in C.

For this entropy metric, there exists a lower and a upper limit given $q = |P|$ and $z = |S|$. Figure 2 (a) visualizes the lower and upper entropy limits for $z = 15$ and $q = [1, 15]$. The lower limit describes the most asymmetric distribution of q property values across all services. For any $q = [1, \dots, c]$ one large group of $z-(q-1)$ services will share the same property value and $q-1$ services will exhibit individual property values. The lower entropy limit PDE_{lower} is defined as:

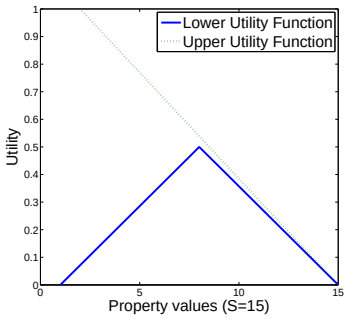
$$\mathrm{PDE}_{lower}(z) = 1 - \frac{q^2 - (2z+1)q + z^2 + z}{z^2 - z} \qquad with\ 1 \le q \le z \tag{2}$$

The upper entropy limit describes the most symmetric distribution of given property values across all services theoretically possible. There exist q groups of $\frac{z}{q}$ services having a distinct property value. The upper entropy limit PDE_{upper} is defined as:

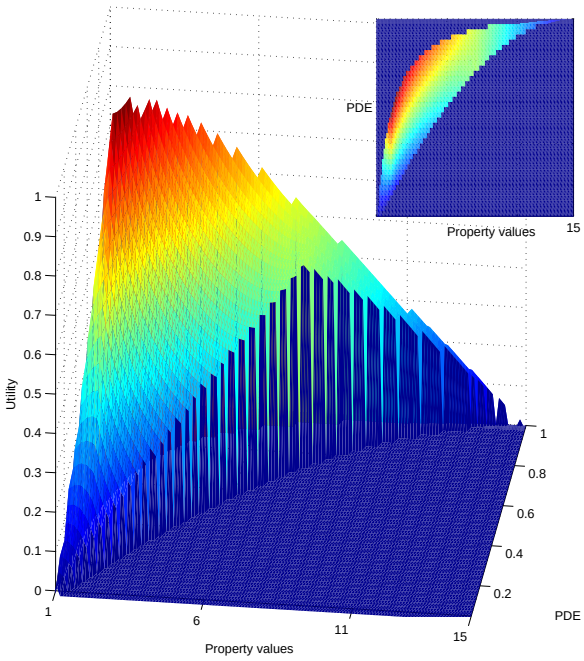
$$\mathrm{PDE}_{upper}(z) = 1 - \frac{(z-q)}{q(z-1)} \qquad with\ 1 \le q \le z \tag{3}$$



(a)



(b)



(c)

Fig. 2. Entropy limits (a), utility boundaries (b), and overall utility function (c) for $c = 15$

The algorithm presented in the next section determines if the impact results in interactions occurring predominantly between services exhibiting the same property values, between services of different property values, or between services without any distinct interaction bias. First, we need to evaluates a property's likelihood of having an impact on interactions.

To this end, we introduce upper and lower entropy utility functions. These utility functions describe the ratio of services that have a choice to communicate either with services of the same property value or with services exhibiting different property values. Only these services generate interactions that exhibit potential property impact [23], [24].

The lower entropy utility function $util_{lower}$ corresponds to the lower entropy limit (PDE_{lower}). It reflects the fact, that $q-1$ services can only communicate with services exhibiting a different property value, and thus cannot be included in the impact calculation. Consequently, as individual property values become more common (i.e., entropy value $\rightarrow$ 1), the likelihood reaches 0. In contrast, as services increasingly share the same property value (i.e., entropy value $\rightarrow$ 0) any interactions across properties must be considered outliers and the likelihood similarly decreases towards 0.

$$\text{util}_{lower}(z) = 0.5 - |-0.5 - \frac{1-q}{z-1}| \qquad 1 < z, 1 < q \leq z \quad (4)$$

The upper entropy utility function $util_{upper}$ corresponds to the upper entropy limit (PDE_{upper}). It peaks where all entities are equally distributed across two property values and decreases steadily as the number of distinct property values rise.

$$\text{util}_{upper}(z) = \frac{z-q}{2-q} \qquad \forall 0 < z, 2 < q \leq z \quad (5)$$

Figure 2 (b) visualizes the lower and upper entropy utility function for $z = 15$ and $q = [1, 15]$. We aggregate upper and lower utility functions in the overall utility function util_{total} defined as follows:

$$\text{util}_{total}(z, pde) = \frac{(pde - PDE_{lower})}{PDE_{upper} - PDE_{lower}} * util_{upper} + \frac{(PDE_{upper} - pde)}{PDE_{upper} - PDE_{lower}} * util_{lower} \quad (6)$$

where $util_{upper}$ returns the utility value for the upper entropy limit, and $util_{lower}$ returns the utility value for the lower entropy limit. The total value combines the two utility values proportional to the distance of the entropy value and the respective upper and lower boundaries (PDE_{upper}, PDE_{lower}). Figure 2 (c) visualizes the overall utility function which provides a likelihood measurement in the interval $[0, 1]$.

V. Property Impact Evaluation Algorithm

The PDE model provides the means to identify promising impact properties. In the subsequent step we need to evaluate whether these candidate properties have indeed an impact on service interactions. We define a positive impact of a property value on a group of services when these services tend to communicate with each other (i.e., internal communication),

rather than interacting with services exhibiting different property values. A negative impact implies a tendency towards external communication.

Algorithm 1: Impact Evaluation Algorithm $\mathcal{A}(G(S,E),P)$

function CalculateImpact($G(S,E),P$)
  $Dev \leftarrow$ call $AnalyzeZeroModel(P,G)$
  for all $Clustersc \in P$ **do**
    $nRatio \leftarrow |c|/|V|$
    $cRatio \leftarrow$ call $CalcLinkRatio(c,G)$
    $diff = |nRatio - cRatio|$
    if $diff * util(P) > 2 * Dev[c]$ **then**
      if $cRatio > nRatio$ **then**
        /* trend for internal communication */
        $trend = diff/(1 - nRatio)$
      else
        /* trend for external communication */
        $trend = diff/nRatio * -1$
      end if
      $setTrend(c, trend)$
    else
      $setTrend(c, 0)$
    end if
  end for
end function

function AnalyzeZeroModel(p,G)
  $Dev[] \leftarrow \emptyset$
  for $i = 1$ *to* d **do**
    $R \leftarrow randomizeAcrossPartitions(G, clusterSizes(P))$
    for all $Clusterr \in R$ **do**
      $nRatio = |r|/|V|$
      $cRatio \leftarrow$ call $CalcLinkRatio(r,G)$
      $diff = |nRatio - cRatio|$
      if $cRatio > nRatio$ **then**
        $dev = diff/(1 - nRatio)$
      else
        $dev = diff/nRatio$
      end if
      $Dev[r] \leftarrow Dev[r] + dev$
    end for
  end for
  for $i = 1$ *to* $|C|$ **do**
    $Dev[i] \leftarrow Dev[i]/z$
  end for
  return Dev
end function

function CalcLinkRatio(c,G)
  $intra = countLinksWithinCluster(c,G)$
  $total = countLinksOfCluster(c,G)$
  $edgeRatio = intra/total$
  return $edgeRatio$
end function

Viable service invocation collection techniques include logging[9], monitoring [10], and sensing [11]. The union of all service invocations create a directed interaction graph $\mathcal{G}(V,E)$. The graph's edges E denote service invocations and the nodes V represent all services $\mathcal{S} \in \mathcal{N}$ participating in the service network. For each property P, we refer to the set of services exhibiting the same property value p as a network *cluster*.

For the impact evaluation process (Algorithm 1), we select properties with highest $util_{total}$. For every cluster, the *cRatio* calculates the ratio of property internal to total communication links. The natural link ratio *nRatio* of a cluster in an unbiased network is $|cluster| \ / \ |S|$. To include the characteristics of the underlying interaction network, we create a zero model by distributing all services randomly across clusters of the same size. Multiple rounds of randomization yield a natural deviation of each cluster ratio from the natural ratio. To enable comparison of clusters independent of their natural ratio (*nRatio*) any deviation from *nRatio* is mapped to the interval $[-1,+1]$, where a $trend_P(c)$ of -1 indicates complete external orientation, and $+1$ complete internal orientation. This orientation is defined as:

$$trend_P(c) = \begin{cases} \frac{cRatio_c - nRatio_c}{1 - nRatio_c} & if\ cRatio_c > nRatio_c, \\ -\frac{cRatio_c - nRatio_c}{nRatio_c} & if\ cRatio_c \leq nRatio_c \end{cases} \quad (7)$$

and the impact of cluster c for Property P is defined as:

$$imp_P(c) = \begin{cases} trend_P(c) & if\ |trend_P(c)| * util_{total}(P) > 2 * dev_c \\ 0 & otherwise \end{cases} \quad (8)$$

where $util_{total}(P)$ is the utility of property P and dev_c is the zero model deviation for the cluster c. Taking twice dev_c and reducing further by $util_{total}(P)$ ensures that also for low likelihood values the deviation is sufficiently distinct.

A property p needs not necessarily consist of uniform cluster trends. Internally oriented, externally oriented, and unbiased clusters can coexist. Aggregating all trends proportionally to their corresponding cluster size yields the overall property importance value:

$$imp_P = \frac{\sum_{i=1}^{n} |imp_P(i)| * |cluster_P(i)|}{|S|} \quad (9)$$

We continue to consider only properties with the highest impact imp_P for further interaction analysis as outlined in the following section.

VI. Service Ranking Algorithm

The calculation and evaluation of property utility, impact, and impact trend is node independent. When a new service joins the network, the ranking algorithm applies these global metrics to generate a recommendation specific to the newcomer. For the properties with highest impact, we select the cluster identified by the newcomer's properties. For each cluster, we derive its interaction affinity towards other clusters. The affinity function $affinity(G, c_1, c_2)$ describes the likelihood of a new request in c_1 being forwarded to c_2. The special

case $c_1 = c_2$ covers internal request delegation. The function is defined as:

$$\text{affinity}(G, c1, c2) = \frac{|links(c1 \rightarrow c2)|}{|links(c1 \rightarrow G)|} \quad (10)$$

where $links(c1 \rightarrow c2)$ selects all links starting in cluster c_1 and ending in cluster c_2, respectively ending anywhere in the network G including c_1. In a directed graph, affinity is not reciprocal, thus $\text{affinity}(G, c1, c2) \neq \text{affinity}(G, c2, c1) \forall c1 \neq c2$.

Our ranking algorithm builds on top of any existing selection mechanism that fulfils following three conditions: (i) returned candidate services are potential communication partners, (ii) services are ranked by their domain specific capability, (iii) services map to ranking scores that reflect the relative match amongst all selected services. A mere list representing the service's rank is insufficient. In case of failing these conditions, our ranking algorithm considers all candidates as equally suitable.

Algorithm 2: Update Ranking Results $\mathcal{A}(new, R, PP)$

function RANKINGRESULTUPDATE(new, R, PP)

/* Modifies the ranking results based on property importance and affinity */

for all *ResultEntry* $r \in R$ **do**

/* affw collects all effects on candidate rank */

$affw = 0$

for all *Property* $P \in PP$ **do**

$c_{newcomer} \leftarrow getCluster(P, new)$

$c_{candidate} \leftarrow getCluster(P, r)$

affinity = calcAffinity($c_{newcomer}, c_{candidate}$)

affw = affw + affinity $*$ $impact(P)$

end for

$updateRank(r, getRank(r) * affw)$

end for

$sort(R)$

end function

The basic idea is to apply cluster affinity values to update the candidate's rank. Algorithm 2 demonstrates the precise steps. For each candidate and all properties of significant impact, as identified in the previous section, we select the newcomer's cluster $c_{newcomer}$ and the candidate's cluster $c_{candidate}$. We subsequently retrieve the affinity value of $c_{newcomer}$ towards $c_{candidate}$. Candidates in clusters with low affinity are penalized more than candidates in clusters of frequent request forwarding. Affinity values do not modify ranks to their full extend but only proportional to the respective property impact $imp_P(c)$. For each candidate the sum of all weighted affinity values determines the extend to which the ranking result is reduced or increased. Finally, the updated candidate list is sorted again. The newcomer service can then select among the top ranked existing services for successful request forwarding.

Recommending services from clusters that have received many requests in previous rounds achieves desirable preferential attachment characteristics. Independent from the number of services, the recommendation algorithm ensures its persisting applicability as the service network grows.

A. Discussion of Computational Complexity

The computational complexity of our approach depends on following factors: the total number of services z, the number of public properties $|\mathcal{P}|$ and the number of their respective values q, the number of service interactions $|E|$, and the number of graph randomizations d. Table II lists the worst case runtime complexity for the various processing steps.

Step	Complexity
Service to Property Mapping	$\mathcal{O}(z * q)$
Entropy Calculation	$\mathcal{O}(\|\mathcal{P}\| * q)$
Interaction to Cluster Mapping	$\mathcal{O}(\|E\| * \|\mathcal{P}\|)$
Cluster Analysis	$\mathcal{O}(\|\mathcal{P}\| * q^2)$
Zero Model Analysis	$\mathcal{O}(\|E\| * \|\mathcal{P}\| * d)$

TABLE II
RUNTIME COMPLEXITY

From this overview, the *Cluster Analysis* appears to inhibit scalability the most. However, by restrict analysis to properties with highest entropy value PDE, the maximum value of observed property values q will grow slower than the number of total services.

VII. EVALUATION AND EXPERIMENTS

This section demonstrates the effectiveness of our approach based on the motivating scenario. This includes a step by step walk-through of metric computation and analysis of multiple properties. We simulate a service network in the second part of this section. The simulation compares the benefit of applying our service ranking algorithm to a trial-and-error selection in terms of accepted and rejected invocations. Throughout the experiment, we measure only the benefit experienced by a newcomer service. In real-world settings, however, any services is free to utilize the ranking algorithm.

A. Scenario

We observe a limited number of services in the network for sake of clarity. The recommendation process observes three public properties: (i) Location (L1…L9), (ii) Organization (O1…O4), and (iii) Capability (C1…C3). Table V (upper part) outlines the mapping of 14 existing services and one newcomer (S15) to the three properties. This configuration yields the property distribution entropy metric (PDE), corresponding entropy limits (PDE_{upper}, PDE_{lower}), and respective utility in Table III.

Analyzing the weighted interaction graph in Table V (lower part), we detect the impact values depicted in Table IV. For *Location* and *Organization*, we derive impact only for L5, respectively O3, in both cases a strong external trend. For *Capability*, the interaction graph results in a strong external trend for all three property values (C1, C2, and C3). Hence,

Property	PDE	PDE_{lower}	PDE_{upper}	$util_{total}$
Loc	0.945	0.835	0.957	0.411
Org	0.802	0.396	0.808	0.826
Cap	0.626	0.275	0.718	0.772

TABLE III
PDE, LIMITS, AND UTILITY VALUES FOR LOCATION, ORGANIZATION, AND CAPABILITY PROPERTIES.

Property										Total
Location	L1	L2	L3	L4	L5	L6	L7	L8	L9	
Impact	0	0	0	0	-1	0	0	0	0	0.07
Organization	O1	O2	O3	O4						
Impact	0	0	-0.95	0						0.20
Capability	C1	C2	C3							
Impact	-0.86	-1	-0.92							0.92

TABLE IV
PROPERTY IMPACT EVALUATION RESULTS

for service S15 with properties (L9, O3, C1) and randomly chosen neighboring services (S2, S4, S7, S9, S11, S12, S14), we arrive at the ranking results printed in the rightmost column of Table V.

Service S2 is ranked highest. As property *Capability* has the strongest impact on the interaction network, we put most weight on affinity values amongst property values C1, C2, and C3. In our scenario, services of type C1 tend to forward requests to service of type C2, C2 to C3, and C3 back to C1. The ranking result thus recommends service S15 to forward requests primarily to S2 as S2 is the only neighbor of S15 exhibiting property C2.

B. Simulation Setup

Simulation-based evaluation allows for analyzing our recommendation algorithm under changing conditions with respect to property count, property impact, service network size, and impact fluctuations. We focus only on the behavioral characteristics of our algorithm and do not consider the costs of network monitoring. Chen et al. [25] follow an algebra-based approach to efficient network monitoring.

The simulation environment consists of $|\mathcal{S}| = n$ services. Each service exhibits $|\mathcal{P}| = m$ property values, corresponding to m distinct properties. Services have the capability to forward a received request to another service from their service neighborhood h or reject it. For each property, an acceptance matrix $\mathcal{M}$ simulates the impact of current requirements on the service interaction structure. The matrix provides the likelihood of any service with property value p_i to accept a request from a service with property value p_j. As the simulation progresses, we adapt the importance weight of the various property matrixes to reproduce the dynamic requirement changes. Table VI provides a snapshot of an acceptance matrix for property *Organization* comprising four property values. In this example, request forwarding occurs in a circle.

from/to	O1	O2	O3	O4
O1	0	1	0	0
O2	0	0	1	0
O3	0	0	0	1
O4	1	0	0	0

TABLE VI
EXAMPLE ACCEPTANCE MATRIX $\mathcal{M}$ FOR FOUR ORGANIZATION PROPERTY VALUES $O1 \ldots O4$ EXHIBITING MAXIMAL CONSTRAINTS.

In each simulation round, services receive r randomly assigned requests. Each service then selects a member from its neighborhood to forward the request to. The receiving service then chooses to accept or deny the request. In the former case, the request is considered successfully completed. In the latter case, the sending service receives 1 penalty point and has to find another service to forward the request to.

Although services apply the acceptance matrix for incoming requests, they do not utilize this information for outgoing requests. Instead, they engage the proposed ranking algorithm. The algorithm then applies the analyzed public properties and service interactions as described in the previous sections. To eliminate any effects of domain specific ranking, the simulation assumes all services are equally able to process a request. We calculate the benefit of our recommendation algorithm by comparing the penalty a newcomer service receives when contacting neighbors by trial-and-error and when contacting the recommended neighbors.

In all experiment iterations, we assign random requests to services each round to simulate service load fluctuations. To keep the overall network load constant, however, the average number of assigned requests per service is fixed at $r = 20$.

C. Measuring Scalability

First, we demonstrate the scalability of our approach. We increase the number of services (n), service neighborhood (h), and property values (p_m). In each round, we measure for each newly added service the penalty received in the process of successfully forwarding a single request to a random neighbor, respectively a recommended neighbor.

The initial service network consists of $n = 50$ services, each having $h = 24$ random neighbors. Four properties (P1 $\ldots$ P4) exhibiting $|p| = 7, 5, 4$, and 4 values respectively exert impact via their acceptance matrixes. As we add a new service, we connect it with random $20 + log(n)^2$ existing services. Additionally, we link random $log(n)$ existing services with the newcomer. For the four properties (P1 $\ldots$ P4), the simulation introduces new property values at a growth rate of $log(n)$.

Figure 3 prints the average benefit for every 50 consecutive benefit measurements over multiple experiment runs. On average, the recommendation-based approach outperforms the trial-and-error approach across scales. At the end of the scalability experiment, the final service network comprises 10050 services, each linked to 105 neighbors on average. Each of the four properties exhibit nine more values, bringing the number of choices to $|p| = 16, 15, 14$, and 13 respectively. The recommendation algorithm yields similar good results for this configuration as for the initial service network.

ServiceId	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15
Location	L1	L2	L1	L2	L3	L4	L5	L6	L7	L8	L9	L4	L8	L9	L9
Organization	O1	O2	O3	O4	O1	O2	O3	O4	O1	O2	O3	O4	O1	O2	O3
Capability	C1	C2	C3	C1	C2	C3	C1	C2	C3	C3	C3	C3	C3	C3	C1

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	Rank
S1	0	0	0	0	35	0	0	0	0	0	0	0	0	0	-
S2	0	0	0	0	0	0	0	0	0	0	0	0	0	43	91.50
S3	33	0	0	6	0	0	0	0	0	0	0	1	0	0	-
S4	0	33	0	0	0	0	0	5	0	0	0	0	0	0	17.77
S5	0	0	0	0	0	0	0	0	0	26	0	0	8	0	-
S6	2	0	0	0	0	0	27	0	0	0	2	0	0	0	-
S7	0	0	0	3	16	1	0	0	1	3	0	0	0	0	3.05
S8	0	0	0	0	0	0	0	0	0	31	1	0	0	0	-
S9	3	0	0	0	0	0	29	0	0	0	0	0	0	0	15.37
S10	0	3	0	28	0	0	0	0	0	0	2	1	2	0	-
S11	0	0	1	32	2	0	0	0	0	0	0	1	0	0	4.95
S12	0	0	0	37	0	0	0	0	0	0	0	0	0	0	13.74
S13	47	0	0	0	0	0	0	0	0	0	0	0	0	0	-
S14	0	0	1	29	2	0	0	0	0	2	0	1	0	0	5.57

TABLE V
SERVICE NETWORK: WEIGHTED DIRECTED GRAPH INCLUDING RANKING RESULTS FOR S15.

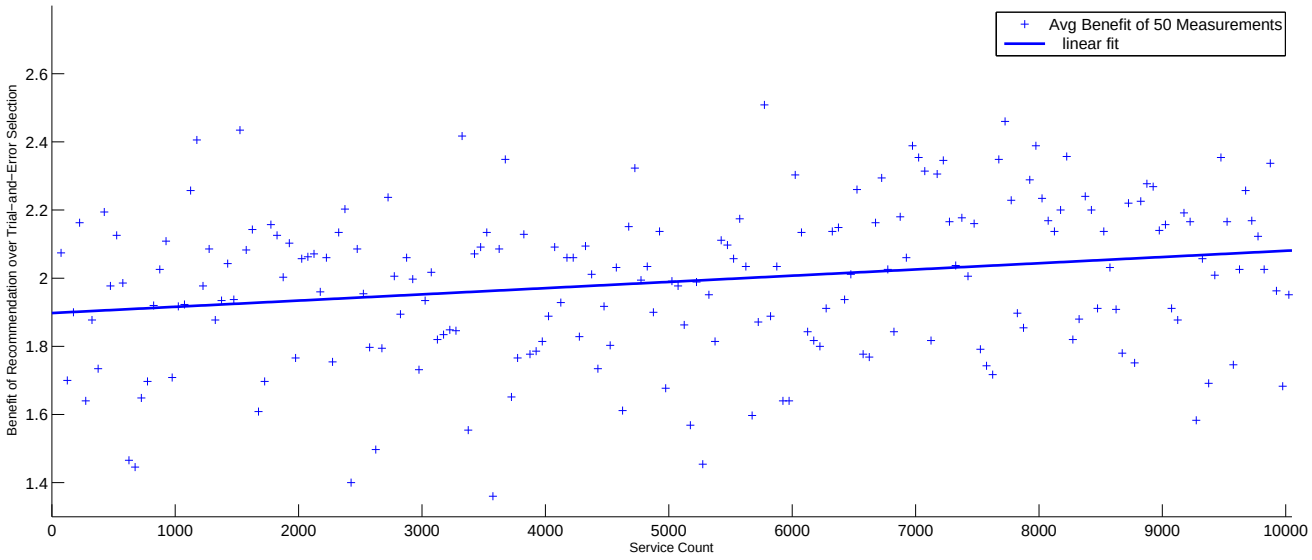


Fig. 3. Average benefit for service recommendation compared to trial-and-error selection. Numbers display aggregation of 50 new services within a service network growing from 50 to 10050 services.

D. Measuring Adaptiveness

We have shown scalability for fixed impact of the four properties (P1 ... P4). Here, we demonstrate the adaptability of our approach. Along these lines, we dynamically change the impact weights of the respective acceptance matrixes ($\mathcal{M}_1 \ldots \mathcal{M}_4$) every 10 rounds while measuring the quality of the recommendation result every round. The number of services $n = 50$, their neighborhood size $h = 24$, and the property values (p_m) remain constant. As we keep the number of services fixed, we select in each round a random existing service to measure the penalties for recommended and trial-and-error neighbor selection.

We analyze 30 experiment iterations, each comprising 100 impact changes. Figure 4 prints the benefit (and standard deviation) received for applying recommended selection for each of the 10 rounds after the property impact change. We observe lower—but still positive—benefit measurements for the first two rounds after a change. As the algorithm self-adjusts, average benefit increases to 2.

E. Measuring Constraint Impact

The realizable benefit heavily depends on the constraints on the service network. When lack of constraints result in high acceptance rates, any random neighbor will most likely

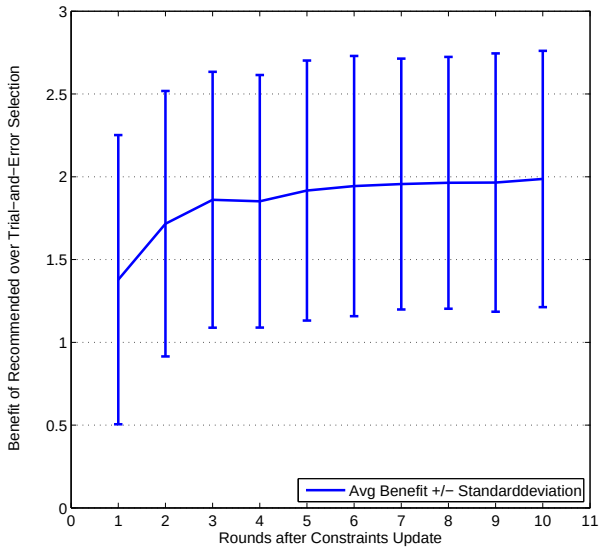


Fig. 4. Average benefit for each round following a property impact change.

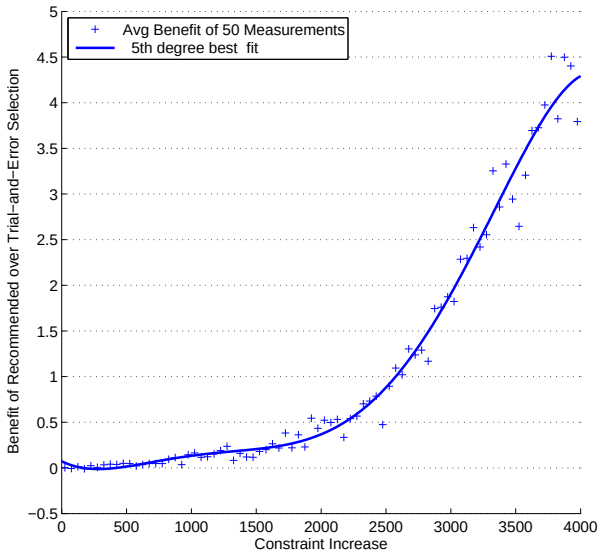


Fig. 5. Average benefit for service recommendation compared to trial-and-error approach for increasing constraints. Numbers display aggregated benefit of 50 consecutive measurements.

be a suitable selection. The ranking algorithm will provide considerable benefit once constraints emerge and begin to increasingly restrict service interactions.

In the third experiment, we start with four properties (each having 10 property values) allowing interactions between any clusters (i.e., the corresponding acceptance matrixes are filled with 1s). Every 10 rounds, we randomly select one particular property and increase the constraints. As we continue to replace random 1s with 0s in the acceptance matrix, the trial-and-error approach yields increasing penalties. We continue increasing the constraints until every acceptance matrix $\mathcal{M}$ contains a single 1 on each row (e.g., Table VI). Thus, for every property P, a service of any particular property value $p_x \in P$ only accepts requests from services exhibiting a single other property value $p_y \in P$ (including $x = y$). Throughout the experiment, property impact and service count remain fixed.

Figure 5 presents the average penalty difference over 10 iterations of $n = 50$ services having on average $h = 24$ neighbors. Benefits start rising around round 1750. Around 3800, this growth levels off as the constraints can no longer be intensified.

F. Experiment Discussion

The simulation reflects the key challenges outlined in the introduction to reproduce the constraints found in real world service networks. First, services provide only public information on their various static properties. Second, the decision process for selecting a suitable receiving service relies purely on dynamic information. Third, services accept incoming requests based only on internal, non-observable information (i.e., defined by the acceptance matrix). Finally, no service obtains a complete view on service interactions.

Despite these challenges, our model and algorithms perform significantly better than trial-and-error service selection. When comparing average, absolute penalty measurements (Figure 6), the ranking algorithm results in 2.5 times lower penalties during the scalability experiment, and 2.7 times lower penalties during the adaptivity experiment, respectively. The third experiment displays 2.1 times lower penalties averaged over the final 1500 rounds. For both scalability and adaptivity experiments, our algorithm requires on average slightly more than a single forwarding retry (i.e., one rejected request). The trial-and-error approach, in contrast, results in approximately three retries. The constraint measurement displays higher failure rates. Our recommendation algorithm requires less than 2.5 retries, while trial-and-error selection causes 5 rejections.

At this stage, we cannot predict the algorithm's performance in real world implementations. However, our simulations yield very promising results and demonstrate both scalability and adaptiveness of our approach.

VIII. Outlook and Conclusions

Self-Organization of services requires knowledge of the significant properties determining service interactions. We have presented a model identifying potential properties. Our impact evaluation algorithm further reduces this set by determining properties with maximum impact on service interactions. Our service recommendation algorithm subsequently ranks suitable request receivers based on the aggregated interaction characteristics. Experimental evaluation has demonstrated both scalability and adaptiveness of our approach. Although focused on data services, our findings apply also to general compositions comprising services and human entities.

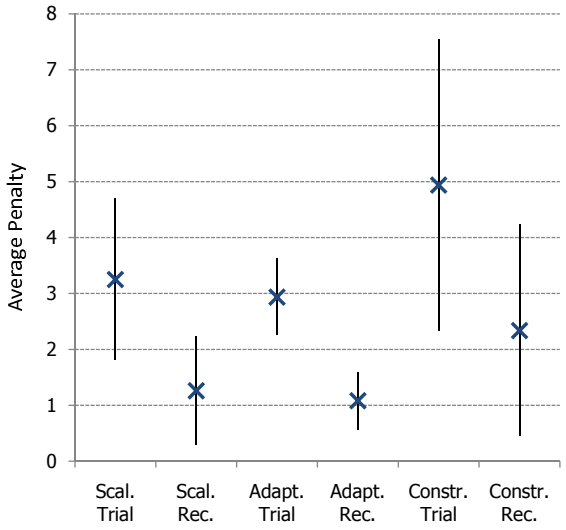


Fig. 6. Average penalty measurements and ± standard deviation for scalability, adaptivity, and constraints experiments; comparing recommended versus trial-and-error selection.

We will continue to aim for enabling services with increasing degree of self-adaptiveness. To this end, we plan extending the set of algorithms utilizing the property distribution entropy. We will also investigate optimal strategies to combine random and ranked selection under varying constraints.

Acknowledgments

This work has been partially supported by the EU STREP project Commius (FP7-213876).

References

[1] M. C. Huebscher and J. A. Mccann, "A survey of autonomic computing—degrees, models, and applications," *ACM Comput. Surv.*, vol. 40, no. 3, pp. 1–28, August 2008. [Online]. Available: http://dx.doi.org/10.1145/1380584.1380585

[2] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, January 2003. [Online]. Available: http://dx.doi.org/10.1109/MC.2003.1160055

[3] S. Hariri, B. Khargharia, H. Chen, J. Yang, Y. Zhang, M. Parashar, and H. Liu, "The autonomic computing paradigm," *Cluster Computing*, vol. 9, no. 1, pp. 5–17, 2006.

[4] R. Quitadamo, F. Zambonelli, and G. Cabri, "The service ecosystem: Dynamic self-aggregation of pervasive communication services," in *Software Engineering for Pervasive Computing Applications, Systems, and Environments, 2007. SEPCASE '07. First International Workshop on*, May 2007, pp. 1–1.

[5] F. Casati, M. Castellanos, U. Dayal, and M.-C. Shan, "Probabilistic, context-sensitive, and goal-oriented service selection," in *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*. New York, NY, USA: ACM, 2004, pp. 316–321.

[6] D. Greenwood and G. Rimassa, "Autonomic goal-oriented business process management," in *ICAS '07: Proceedings of the Third International Conference on Autonomic and Autonomous Systems*. Washington, DC, USA: IEEE Computer Society, 2007, p. 43.

[7] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl, "A journey to highly dynamic, self-adaptive service-based applications," *Automated Software Engg.*, vol. 15, no. 3-4, pp. 313–341, 2008.

[8] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, and T. Urnes, "Design patterns from biology for distributed computing," *ACM Trans. Auton. Adapt. Syst.*, vol. 1, no. 1, pp. 26–66, 2006.

[9] C. Dorn, H.-L. Truong, and S. Dustdar, "Measuring and analyzing emerging properties for autonomic collaboration service adaptation," in *ATC '08: Proceedings of the 5th international conference on Autonomic and Trusted Computing*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 162–176.

[10] O. Moser, F. Rosenberg, and S. Dustdar, "Non-intrusive monitoring and service adaptation for ws-bpel," in *WWW '08: Proceeding of the 17th international conference on World Wide Web*. New York, NY, USA: ACM, 2008, pp. 815–824.

[11] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *Communications Magazine, IEEE*, vol. 40, no. 8, pp. 102–114, Aug 2002.

[12] M. Andreolini, S. Casolari, and M. Colajanni, "Autonomic request management algorithms for geographically distributed internet-based systems," in *Self-Adaptive and Self-Organizing Systems, 2008. SASO '08. Second IEEE International Conference on*, Oct. 2008, pp. 171–180.

[13] E. M. Maximilien and M. P. Singh, "Toward autonomic web services trust and selection," in *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*. New York, NY, USA: ACM, 2004, pp. 212–221.

[14] A. Bottaro and R. Hall, *Dynamic Contextual Service Ranking*, ser. Lecture Notes in Computer Science. Springer, 2007, ch. Software Composition, pp. 129–143.

[15] A. Colman, "Exogeneous management in autonomic service compositions," in *ICAS '07: Proceedings of the Third International Conference on Autonomic and Autonomous Systems*. Washington, DC, USA: IEEE Computer Society, 2007, p. 25.

[16] B. Jennings, S. van der Meer, S. Balasubramaniam, D. Botvich, M. Foghlu, W. Donnelly, and J. Strassner, "Towards autonomic management of communications networks," *Communications Magazine, IEEE*, vol. 45, no. 10, pp. 112–121, October 2007.

[17] T. Yu and K.-J. Lin, "Adaptive algorithms for finding replacement services in autonomic distributed business processes," in *Autonomous Decentralized Systems, 2005. ISADS 2005. Proceedings*, April 2005, pp. 427–434.

[18] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, "A survey of autonomic communications," *ACM Trans. Auton. Adapt. Syst.*, vol. 1, no. 2, pp. 223–259, 2006.

[19] V. Bryl and P. Giorgini, "Self-configuring socio-technical systems: Redesign at runtime," *ITSSA*, vol. 2, no. 1, pp. 31–40, 2006.

[20] F. Saffre, R. Tateson, J. Halloy, M. Shackleton, and J. L. Deneubourg, "Aggregation Dynamics in Overlay Networks and Their Implications for Self-Organized Distributed Applications," *The Computer Journal*, 2008. [Online]. Available: http://comjnl.oxfordjournals.org/cgi/content/abstract/bxn017v1

[21] J. Zhang and R. Figueiredo, "Autonomic feature selection for application classification," in *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, June 2006, pp. 43–52.

[22] D. Marinescu, J. Morrison, C. Yu, C. Norvik, and H. Siegel, "A self-organization model for complex computing and communication systems," in *Self-Adaptive and Self-Organizing Systems, 2008. SASO '08. Second IEEE International Conference on*, Oct. 2008, pp. 149–158.

[23] B. Bollobas, *Random Graphs*, W. Fulton, A. Katok, F. Kirwan, P. Sarnak, B. Simon, and B. Totaro, Eds. Cambridge University Press, 2001.

[24] I. A. Mcculloh, J. Lospinoso, and K. Carley, "Social network probability mechanics," in *MATH'07: Proceedings of the 12th WSEAS International Conference on Applied Mathematics*. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS), 2007, pp. 319–323.

[25] Y. Chen, D. Bindel, H. H. Song, and R. H. Katz, "Algebra-based scalable overlay network monitoring: algorithms, evaluation, and applications," *IEEE/ACM Trans. Netw.*, vol. 15, no. 5, pp. 1084–1097, 2007.