Scalable Computing: Practice and Experience

Scientific International Journal for Parallel and Distributed Computing

ISSN: 1895-1767

Universitatea de Vest din Timişoara

Volume 16(3)

September 2015

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 16, Number 3, September 2015

TABLE OF CONTENTS

DOI 10.12694/scpe.v16i3.1097
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 16, Number 3, pp. iii–iv. http://www.scpe.org © 2015 SCPE

INTRODUCTION TO THE SPECIAL ISSUE ON COLLECTIVE ADAPTIVE SYSTEMS

Collective Adaptive Systems (CAS) is a broad term that describes large scale systems that comprise of many units/nodes, each of which may have their own individual properties, objectives and actions. Decision-making in such a system is distributed and possibly highly dispersed, and interaction between the units may lead to the emergence of unexpected phenomena. CASs are open, in that nodes may enter or leave the collective at any time, and boundaries between CASs are fluid. The units can be highly heterogeneous (computers, robots, agents, devices, biological entities, etc.), each operating at different temporal and spatial scales, and having different (potentially conflicting) objectives and goals, even if often the system has a global goal that is pursued by means of collective actions. Our society increasingly depends on such systems, in which collections of heterogeneous technological nodes are tightly entangled with human and social structures to form artificial societies. Yet, to properly exploit them, we need to develop a deeper scientific understanding of the principles by which they operate, in order to better design them.

The aim of this special issue is to provide a selection of the state of the art, emerging trends, new technologies and best practices in the field of collective adaptive systems. The idea was born at the second FoCAS Workshop on Fundamentals of Collective Adaptive Systems at SASO 2014 in London, however, an open call enabled any researcher working on a related topic to submit a paper for review.

This special issue features four articles that concern collective adaptive systems.

The first one, "On Intention-Propagation-Based Prediction in Autonomously Self-adapting Navigation", by László Z. Varga considers road-traffic systems as an example of a collective adaptive system, and applies intention-propagation-based approaches to forecast travel times. The authors find that using intention-propagation-based traffic forecast provides an important improvement to a simple naive online strategy, and also point out some surprising results; for example, if vehicles use a simple naive strategy and exploit intention-propagation-based prediction, then in some networks and in some cases the traffic may be worse off by exploiting real-time information than without exploiting real-time information. They also suggest other collective adaptive systems to which the techniques could be applied, for instance manufacturing and cloud computing.

The second one, "Application of collective movement in real life scenarios: overview of current flocking solutions", by Bernát Wiandt, András Kökuti, Vilmos Simon considers collective movement of dynamic nodes within an adaptive system, addressing the issue of how to control the nodes in such a way that they behave as a coherent group (flock). The authors critically survey the existing flocking algorithms, showing that the majority are unsuitable for real world use due to overlooking crucial physical limitations or lack of sufficiently detailed models. The paper provides clear directions for future work based on a comprehensive survey and analysis of real-world scenarios.

The third paper, "Overlay Service Computing - Modular and Reconfigurable Collective Adaptive Systems", by Evangelos Pournaras, addresses the design of overlay layers in network services that support collective adaptive systems. The author introduces the notion of overlay services that provide generic application capabilities of a broad application scope enabled by one or more overlay networks. In the paper the ASMA architecture is presented, which supports the definition of structures of overlay services and guides their development by means of a simple high-level notation. The paper reports also two examples of overlay services developed according to ASMA.

The fourth paper, "SODAP: Self-Organized Topology Protection For Superpeer P2P Networks", by Paul L. Snyder and Giuseppe Valetto, addresses peer-to-peer systems organized in superpeer overlays. To address the vulnerability problems of the superpeers, the authors propose SODAP (Self-Organized Degree Adaptation Protection), which is a self-organized strategy for the self-protection of the overlay. The proposed approach is based on the local adaptation of a peer's degree in response to disconnections, by creating redundant connections that lead to a more resilient topology of the system. The authors provide an evaluation of the proposed approach by means of Myconet, a self-organized superpeer overlay for unstructured peer-to-peer networks.

We would like to thank the editorial board of SCPE for the chance of arranging this special issue, and all the reviewers for their hard work.

Giacomo Cabri and Emma Hart

DOI 10.12694/scpe.v16i3.1098
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 16, Number 3, pp. 221–232. http://www.scpe.org © 2015 SCPE

ON INTENTION-PROPAGATION-BASED PREDICTION IN AUTONOMOUSLY SELF-ADAPTING NAVIGATION [*]

LÁSZLÓ Z. VARGA[†]

Abstract. It is widely believed that road traffic as a whole self-adapts to the current situation to make travel times shorter by avoiding congestions, if the autonomously operating navigation devices exploit real-time traffic information. The classical theoretical models do not have definite answer if car navigation based on real-time data is able to self-adapt and produce better traffic, or not. The novel theoretical approach to study this belief is the online routing game model. Current commercial car navigation systems are modelled with the class of simple naive online routing games. It is already proved that simple naive online routing games may show undesirable phenomena. One of the approaches to improve car navigation is intention-propagation-based prediction, where agents share their intention, and can forecast future travel times. In this paper we prove that in spite of exploiting this type of prediction in online routing games, the phenomena studied in simple naive online routing games are still possible, although in a different way. With these theoretical results we point out hitherto hidden pivotal phenomena of intention-propagation-based prediction in collective adaptive systems.

Key words: online routing games, benefit of online data, prediction

AMS subject classifications. 68T42, 91A40, 93C40, 94A15

1. Introduction.

Current road traffic navigation systems have access to real-time data, and they can be regarded as autonomously self-adapting navigation systems. They are *autonomous*, because everyday traffic is not coordinated by a centralized system, and the traffic participants make their own autonomous decisions based on their intentions and the information available for them locally. This means that instead of centralized decision making, we have a *collective* of autonomous distributed decision makers, i.e. a multi-agent system. In this aspect autonomy refers to the autonomous route planning by the navigation devices in the individual cars, instead of following the instructions of some centralized planner. Even if the traffic was coordinated by a centralized system, there would be the question whether the individual traffic participants would conform to its instructions. Current navigation systems are *self-adapting* as well, because the navigation devices in the cars can get up-to-date information on the current status of the traffic, like the current travel time on each road, indicating the current situation of the traffic that needs to be adapted to. The routing algorithms implemented in the navigation devices must be able to utilize this real-time data to self-heal the global traffic, for example if a road becomes congested then the navigation devices autonomously tell the individual cars how to adapt to the current traffic situation, and send the cars to less congested roads. Ultimately, road traffic guided by navigation devices that utilize real-time data is a *collective adaptive system*.

Although current navigation systems are already able to utilize real-time data for route planning, these systems were implemented without clearly understanding how real-time data impacts traffic as a whole, and how real-time data affects the above mentioned autonomous and self-adaptation aspects of navigation systems. Note that in this paper we focus on online-data-based self-adaptation. This is different from self-adaptation based on previous experiences, like, for example, in the case of route selection from home to work based on the experience of the previous day.

Autonomous agents of car navigation devices have access to online real-time data, and they create plans how to achieve their goals in an environment where they jointly utilize resources that become more costly as more agents use them. For example, the online data are the current travel times on roads, the goal is to get to a destination, the created plan is the travel route from the starting point to the destination, the jointly utilized resources are the roads, the cost of resource utilization is the travel time on the road, and the travel time on a road increases when more cars are on that road. Agents are dynamically arriving and departing. The plans are created by exploiting online data that describe the current status and the current cost of the resources. There

[*]Research supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TÁMOP-4.2.2.C-11/1/KONV-2012-0013), and is part of TUD COST Action TU1102 Autonomic Road Transport Support Systems.

[†]ELTE Regional Knowledge Centre, Irányi Dániel u. 4., Székesfehérvár, H-8000 Hungary (http://people.inf.elte.hu/lzvarga/).

is uncertainty about the feasible decision of an agent, because the cost of the resources will change by the time the agent starts to use them: departing agents will release the resources, agents simultaneously creating their plans will influence each others' costs, and agents arriving later may also influence the costs of the resources used by agents already executing their plans.

We do not know much about the benefit of real time information, not to mention how to optimize route planning based on real-time data to make the traffic self-adapt to the current situation, and self-heal itself. The novel *online routing game* model [1] is a theoretical model to formally investigate if autonomously self-adapting car navigation produces better traffic, or not. There is need for such theoretical studies, because driverless autonomous cars are being designed, and online navigation devices based on the (possibly multiple objective [2]) shortest path search strategy are widely installed in many cars. Shortest path search is basically the same as the simple naive strategy that was discussed in [1], and it was proved that if car navigation devices use shortest path search, and exploit real-time data, then the traffic may show undesirable phenomena.

In order to improve the simple naive strategy, the approach of intention propagation was proposed in the anticipatory vehicle routing system using delegate multi-agent systems [3]. In this paper we take the online routing game model of [1], refine it, and then we extend this model to formally investigate and prove some of the properties of the usage of intention-propagation-based prediction in car navigation systems. These investigations are on the formal model level, so the results are valid for any online routing game using intention-propagation-based prediction, including real-time data based car navigation and traffic. The intention-propagation-based prediction can be implemented with or without bio-inspired design patterns [17] as long as they conform to the theoretical model. We do not deal with the implementation of intention-propagation-based prediction, therefore we are not investigating if its implementation can scale with the number of nodes, which may be hundreds or thousands in practice. We assume that intention-propagation-based prediction is somehow implemented, and we are investigating its benefits on the theoretical level.

In the next section we discuss related work and how we advance the state of the art, in Sect. 3 we shortly explain and refine the online routing game model, in Sect. 4 we summarise the intention-propagation-based prediction approach, in Sect. 5 we prove four properties of online routing games which exploit intention-propagation-based prediction, and finally in Sect. 6 we conclude our work.

2. Related Work. Several authors investigated with simulation tools how the traffic would self-adapt if the majority of vehicles used online traffic information for route planning, and they concluded that online data has to be used carefully. A simple scenario consisting of two parallel routes was investigated in [4], and the simulations showed that online information often leads to oscillations in the number of cars on the routes, the velocity, and travel times, which lead to worse overall performance. In the discussion, the authors conclude that one of the reasons for the oscillations is that the real-time information reflects the state of the network some time ago. Another reason for the oscillation is that the agents do not coordinate their actions. In order to improve these, the authors advise the usage of anticipatory traffic forecast based on the broadcast route choice of the agents, which basically means that the agents share or propagate their intentions.

Anticipation [14] is essential to multi-agent systems in order to be able to behave proactively, and adapt to changing situations. Several design patterns were studied [15] to help the systematic design of self-organising emergent systems that show anticipatory behaviour. Many of these design patterns are inspired by biological systems [16], which are well suited to design pervasive service ecosystems like crowd steering [18]. Among the bio-inspired design patterns [17], the digital pheromone pattern and the evaporation pattern are the two most relevant to implement intention propagation. The online routing game model of this paper abstracts away from these design patterns, and models the mechanisms of these design patterns with the ability of the agents of the self-organising system to spread their truthful intentions, and the ability of the infrastructure layer to compute the future status of the system as a result of the intended actions of the agents (see Sect. 4 and the beginning of Sect. 5).

An empirical intention propagation system using such design patterns was elaborated in the anticipatory vehicle routing simulation system using delegate multi-agent systems [3]. This simulated distributed delegate multi-agent system is able to forecast future traffic conditions based on the broadcast intention of the vehicle agents. This forecast information is then used by the vehicle agent to make routing decisions. The approach was evaluated in a simulated network of Leuven, using a statistical approach, and the authors concluded that the use of forecast data results in shorter trip durations than the use of real-time data from Traffic Message Channel (TMC). The statistical approach took into account the trip durations from the anticipatory route planning and the mean values of a perceived distribution. The extreme values of trip durations were not included in the evaluation. The authors claim that the forecast data not only allow drivers to avoid existing congestions, but prevents them from forming congestion as well. The authors admit that providing efficient and stable routes to vehicles will be challenging, even with the use of intention propagation.

Simulations are good for initial evaluations of the approaches, but we can expect a complete evaluation only from theoretical models. The most relevant model is algorithmic game theory, which studies networks with source routing (Sect. 18 in [5]), where end users simultaneously choose full routes for their traffic, and the traffic is routed in a congestion sensitive manner. The algorithmic game theory investigations of the routing game problem revealed important basic properties, like the conditions for the existence of equilibrium, or the limit on the "price of anarchy". However the algorithmic game theory model contains assumptions which do not hold in traffic routing using real-time data. These assumptions are: a) the throughput characteristic of the network does not change with time, and the agents know the characteristic functions; b) the agents simultaneously decide their optimal route; and c) the outcome travel time for a given agent depends on the choice of all the agents and the characteristic of the network, but not on the trip schedule of the other agents.

The issue of traffic dynamism is studied in the field of dynamic traffic assignment [6], but there they investigate the time-varying properties of traffic flow, whereas here we assume that the traffic flow is constant, and only the cost functions may change. In our investigations the critical issue is the sequential decision making of the autonomous agents, which self-adapt to the observed real-time status of the world. This is partly handled by online mechanism design [9].

The novel online routing game model [1] has less restrictive assumptions than the routing game model, and the online routing game approach more realistically models traffic routing based on real-time data by combining the algorithmic game theory approach with the approaches of online algorithms [10] and online mechanism design. Online routing games are different from resource allocation or minority games [11], which are simultaneous one shot or repeated simultaneous games where there might be some coordination among some of the agents. In contrast, online routing games are continuous and non-cooperative games exploiting real-time data. Online routing games are classified [1] into different classes depending on the decision strategies used by the agents. Commercial car navigation software use shortest path search in the road network, possibly taking into account real time traffic data and modifying the static data with the online information about the actual traffic delays. This decision strategy is the simple naive strategy studied theoretically in [1]. It was proved that stability is not guaranteed in the class of simple naive online routing games, and it may happen in some networks that a single flow enters the network, and on some edge inside the network the flow is bigger than the one that entered the network. It was proved that the worst case benefit of online data in some networks may be bigger than one, which means that the worst case travel time with online data may be more than the travel time without online data.

The online routing game model allows the agents to exploit the services of an "advisory" like the intention-propagation-based forecast of the delegate multi-agent system in [3] or the central planner in the PLANETS system [12]. In this paper we advance the state of the art by formally investigating the usage of intention-propagation-based prediction in online routing games, and proving four properties of the class of simple naive intention propagation online routing games. Specifically we prove that in spite of using intention-propagation-based prediction, in some online routing games a single flow that entered the network may become bigger on some edge inside the network, the traffic may fluctuate, the worst case benefit of online data may be bigger than one, i.e. it may be a "price", and this price may be arbitrarily large. By these theoretical results, we pinpoint the front lines of future research on self-adapting autonomous traffic systems.

3. Formal Analysis with Online Routing Games. The online routing game model of [1] is a combination of the algorithmic game theory model of the routing problem and the online mechanisms. The online routing game model resembles the algorithmic game theory routing game model in the concepts of flow, cost, and resources, and it resembles the model of online mechanisms in the sequences of time periods and decisions. A T time unit is introduced, because the costs of the resources depend on the rate of their utilization. The online routing game model of [1] needs some additional refinements, which were not relevant in [1], but are needed here, so we refine the model of [1] in the following. The online routing game model is the sextuple (t, T, G, c, r, k), where

- $t = \{1, 2, ...\}$ is a sequence of equal time periods.
- T is a natural number, and T time periods give one time unit (e.g. one minute).
- G is a directed multi-graph[1] $G = (V, E)$ with vertex set V and edge set E where each $e \in E$ is characterized by a cost function c_e and a a time gap gap_e.
- c is the cost function of G with $c_e : R^+ \rightarrow R^+$ for each edge e of G, and it maps the flow[2] $f_e(\tau)$ (that enters the edge e at time τ) to the travel time on the edge. c_e for the agent entering the edge e is never less than the remaining cost of any other agent already utilizing that edge increased with a time gap[3] gap_e, which is specific to edge e. The value of the flow $f_e(\tau)$ is the number of agents that entered the edge e between $\tau - T$ (inclusive) and τ (non-inclusive). If two agents enter an edge exactly at the same time τ, then one of them (randomly selected) suffers a delay gap_e, which is part of its cost on edge e, and its remaining cost is determined at the delayed time, so its cost on edge e will be $gap_e + c_e(f_e(\tau + gap_e))$. The cost functions are nonnegative, continuous, and nondecreasing. The cost functions have a constant part which does not depend on the flow on the edge, and a variable part which depends on the flow on the edge. The variable part is not known to any of the actors of the model until an agent exits an edge, and reports it.
- r is the total flow given by a vector of flows with r_i denoting the flow aiming for a trip P_i from a source vertex s_i of G to a target vertex t_i of G. The flow r_i is given by $T \div n_i$ where $n_i \in [1, \ldots, T]$ is a natural number constant, meaning that the following distance of the agents of the flow r_i are n_i time periods. Thus the value of r_i is between 1 and T. (Note that T, and the time unit to be modelled by T, have to be selected according to the flow values to be modelled.)
- $k = (k^1, k^2, ...)$ is a sequence of decision vectors with decision vector $k^t = (k^t_1, k^t_2, ...)$ made in time period t. k^t_i is the decision made by the agent of the flow r_i in time period t. The decision k^t_i is how the trip P_i is routed on a single path of the paths leading from s_i to t_i. The actual cost of a path $(e_1, e_2, e_3, ...)$ for a flow starting at time τ is $c_{e1}(f_{e1}(\tau)) + c_{e2}(f_{e2}(\tau + c_{e1}(f_{e1}(\tau)))) + c_{e3}(f_{e3}(\tau + c_{e1}(f_{e1}(\tau)) + c_{e2}(f_{e2}(\tau + c_{e1}(f_{e1}(\tau)))))) + ...$, because the actual cost of an edge is determined at the time when the flow enters the edge.

The actual cost of the edges become known for the agents in the system only when an agent reports its actual cost. Because agents do not report cost values in each time step, the agents interested in the cost values must decrease the last reported value by taking into account the time elapsed since the last reporting event (it is similar to the pheromone evaporation in [13]).

With the online routing game model, we can investigate formally if the agents are better off by making decisions based on online real-time data, or not. In the algorithmic game theory model, if there is equilibrium then the price of anarchy is the ratio between the equilibrium and the optimum. It was proved in [1] that equilibrium is not guaranteed in simple naive online routing games, therefore the equilibrium cannot be used to define the benefit of real-time data in online routing games. In order to be able to compare the costs of the agent system using online data with the costs of the agent system not using online data, three kinds of benefit of online data were defined [1] for the worst, average, and best cases.

Definition 3.1. *The worst(/average/best) case benefit of online real time data at a given flow is the ratio between the cost of the maximum(/average/minimum, correspondingly) cost of the flow and the cost of the same flow with an oracle using the same decision making strategy with only the constant part of the cost functions.*

The agents are happy with online-data if the benefit values are below one. If the worst case benefit of online data is above 1, then the agents may be sometimes worse off by utilizing online data than without online data. If the best case benefit of online data is above 1, then the agents are always worse off by utilizing online data than without online data.

It is important to note here that most of the reported empirical evaluation of the different navigation systems based on real-time data focused on the average travel times, with an analysis of the distribution (for

[1] In this model the graph G may contain parallel edges.
[2] The flow consists of units, called agents, that enter the network in one of the time periods.
[3] In this model cars cannot overtake the cars already on the road and there is a time gap, i.e. minimum "following distance".

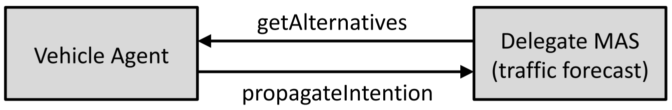


FIG. 4.1. *Intention propagation: the vehicle agent asks from the delegate MAS the travel time forecast on different routes, selects the best route, and sends back its intended route to the delegate MAS.*

example [3]), but did not include the determination of the maximum travel time. In our view this approach is good from the statistical point of view, but the users of a navigation system based on real-time data would be interested in the extreme values as well. In a practical setting, if a user experiences at least once that the travel time based on real-time data is, lets say, ten times longer than the travel time without real-time data, then, in our view, this user would never use the navigation system based on real-time data again. Therefore the evaluation of the navigation systems should include the worst case benefit defined in [1]. As we mentioned earlier, the formal analysis of the class of simple naive online routing games proved that the users of the typical commercial navigation systems in some networks may be sometimes worse off by using real-time data than by not using real-time data.

4. Intention-Propagation-Based Prediction.

As it was noted in [3] and [4], one of the disadvantages of the class of simple naive online routing games is that they react upon traffic congestions only after the congestions have occurred, and they do not try to prevent them. In order to alleviate this problem, anticipatory [14] vehicle routing is implemented in the simulation environment of [3] which uses the individual planned routes of the agents to forecast future traffic density. Every vehicle is represented by a vehicle agent running on a smart device inside the vehicle. Vehicle agents communicate with the delegate multi-agent system (delegate MAS). The delegate MAS is implemented with the usage of different bio-inspired design patterns [17]. The delegate MAS represents the traffic environment, and it is able to make forecast of future traffic density based on the current traffic situation and the planned routes of the vehicles. The delegate MAS provides the traffic forecast back to the vehicle agents, which use this information to plan their trip. Fig. 4.1 shows how the delegate MAS and the vehicle agents interact in this intention propagation process. We are using this abstraction in our model, because it is independent from the actual implementation of the traffic forecaster as long as the traffic forecaster computes the forecasts in accordance with the planned routes of the vehicles.

The delegate MAS of [3] can predict future travel times based on the intention notifications that it has received from all vehicle agents. The delegate MAS has a parametrised model that describes the relationship between the travel time and the intention notifications. The parameters are continuously updated, based on both historical and real-time data, so basically the delegate MAS computes the cost functions of the online routing game model with the ability to handle varying cost functions.

If the predicted future travel times show that a new travel route is preferable, then the vehicle agent is free to change its route plan. If the vehicle agent changes its route plan, then it notifies the delegate MAS of its change of intention. The old intention is then invalidated, and the new intention is registered in the delegate MAS.

Although the vehicle agent could use several strategies to revise its intention, the decision strategy is not detailed in [3], so we assume that vehicle agents always select the shortest travel time. This route selection strategy is called simple naive decision strategy in the online routing game model.

5. Intention-Propagation-Based Prediction in Online Routing Games.

Now we are going to use the above anticipatory vehicle routing system to define and formally analyse the class of online routing games that use intention-propagation-based prediction in their decision mechanism. We call this class of online routing games simple naive intention propagation online routing games.

DEFINITION 5.1. *Simple naive intention propagation online routing games (SNIP online routing games) are online routing games where*

- *the decision making agents of the flows are the vehicle agents of the anticipatory vehicle routing system,*
- *the delegate MAS of the anticipatory vehicle routing system predicts the travel times for each path of the trip P_i, based on the planned routes of the agents currently in the system,*

- *and the decision k_i^t is to select the path with the shortest travel time among the predicted travel times on the different paths of the trip P_i.*

The vehicle agent notifies the delegate MAS of its selected path, and the delegate MAS remembers this selection while the vehicle agent is in the road network, and the delegate MAS invalidates it when the vehicle agent exits the network.

Note that SNIP online routing games are a little bit different from the anticipatory vehicle routing system of [3], because the SNIP vehicle agents select their route when they enter the road network, and, in accordance with the online routing game model, they do not revise it during their trip. In the future, we may develop a dynamic version of the online routing game model to be able to model dynamic re-routing. The reasoning of the SNIP online routing game agent is thus the following:

Reasoning of the vehicle agent in the SNIP online routing game

```
1: // Get the predicted travel times
2: routes := getAlternatives(source,target)
3: // Select the shortest path
4: selectedRoute := choose_shortest(routes)
5: intention := selectedRoute
6: // Propagate the current intention
7: propagateIntention(intention)
8: // Instruct the driver to destination
9: instructDriver(intention)
```

We assume that agents signal their intentions truthfully. We do not know if truthfulness is individually rational or not, but in the future, the SNIP online routing game model may be used to investigate this question as well. At this time, we want to investigate if the additional information about the cognitive state of the agents already in the network helps, or not. When the agents enter the network of the SNIP online routing game, they base their decisions not only on the current status of the network (the travel times on roads), but also on the current cognitive status of the agents (the intentions of the agents) already in the network.

We investigate the class of SNIP online routing games with formal methods to see if the properties of the SNIP online routing games are different from the properties of the simple naive online routing games, or not. In SNIP online routing games the agents receive a prediction of future traffic, so we would expect that this additional information can be used to improve the properties of simple naive online routing games. Unfortunately intention-propagation-based prediction does not eliminate fully the "single flow intensification" problem, as the next Theorem shows.

Theorem 5.1. *There are simple naive intention propagation online routing games, where the total traffic flow has only one incoming flow, i.e. $r = (r_1)$, however the flow on some of the edges sometimes may be more than r_1.*

Proof. We are going to show how this happens in the SNIP online routing game $SN_1 = (t, T, G, r, c, k)$ which is described as follows. $T = 10$, modelling one time unit[4]. The network G is given as a three node directed graph with $V = \{v_1, v_2, v_3\}$ and $E = \{e_1, e_2, e_3\}$ as shown in Fig. 5.1. Note that the edges e_1 and e_2 are parallel, but have different cost functions. The cost function c of the road network is $c_{e1} = 10 + x$, $c_{e2} = 10.5 + x$, and $c_{e3} = 1 + x$, where x is the total incoming flow on the edge. The total traffic flow is $r = (r_1)$ with $r_1 = 1$ on the trip P_1 from the source vertex v_1 to the target vertex v_3 of G. The trip P_1 has the set of paths $\{p_1, p_2\}$, where $p_1 = (e_1, e_3)$ and $p_2 = (e_2, e_3)$.

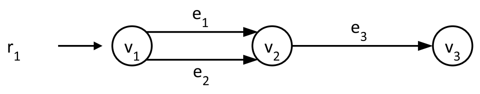


Fig. 5.1. *The network of the SNIP online routing game SN_1.*

[4]In this case, the time unit consists of 10 time steps, so time 1 is at time step 10.

Table 5.1
Predictions, path selections and actual travel times in SN_1.

Start at v_1	Path	Predicted at v_2	Predicted at v_3	Selected path	Actual at v_3	Travel time
0	(e_1, e_3)	**10**	11	(e_1, e_3)	**11**	11
0	(e_2, e_3)	10.5	11.5			
1	(e_1, e_3)	12	13			
1	(e_2, e_3)	**11.5**	12.5	(e_2, e_3)	**12.5**	11.5
2	(e_1, e_3)	**12**	14	(e_1, e_3)	**14**	12
2	(e_2, e_3)	13.5	14.5			
3	(e_1, e_3)	14	15			
3	(e_2, e_3)	**13.5**	14.5	(e_2, e_3)	**14.5**	11.5
4	(e_1, e_3)	**14**	16	(e_1, e_3)	**16**	12
4	(e_2, e_3)	15,5	16.5			
...	...	...	...	...	...	...

Table 5.1 shows the predictions, the path selections and the actual travel times for the first five agents of the flow r_1. The first column of the table is the start time of the given agent of r_1. Because $r_1 = 1$, one agent starts at the beginning of each time unit. The second column is the path for which the predictions are shown in the third and fourth column. The third column shows the predicted arrival time at vertex v_2 and the fourth column shows the predicted arrival time at vertex v_3. The fifth column shows the path selected by the given agent of r_1 based on the prediction. The sixth column shows the actual arrival time at vertex v_3, and the last column shows the actual travel time of the given agent. The boldface figures show the actual arrival times. The actual arrival time at vertex v_2 must always be the same as one of the predicted arrival times, because vertex v_2 is always the end vertex of the first edge of the path. This is why the actual arrival time at vertex v_2 is boldfaced in the third column.

The predictions for the agent starting at time 0 are based on the fixed parts of the cost functions, because the edges are empty. So the predicted travel time on e_1 is 10, on e_2 it is 10.5, and on e_3 it is 1. Thus the predicted arrival times at vertex v_2 are 10 on path (e_1, e_3) and 10.5 on path (e_2, e_3). The predicted arrival times at vertex v_3 are 11 on path (e_1, e_3), and 11.5 on path (e_2, e_3). The predicted fastest path is (e_1, e_3), so this agent of the flow selects this path. The next agent of the flow starts at time 1, and it cannot enter e_3 before time 11, so the agents starting at time 1 and later cannot interfere with the agent starting at time 0, and the actual arrival times of the agent starting at time 0 are the same as the predicted arrival times on path (e_1, e_3).

The predictions for the agent starting at time 1 are more complex, because the previous agent influences the travel time on one of the roads. The predicted travel time on e_1 is 11, because the agent starting at time 0 entered this road 1 time unit before. The predicted travel time on e_2 is 10.5, because this road is empty. Thus the predicted arrival times at vertex v_2 are time 12 on path (e_1, e_3) and time 11.5 on path (e_2, e_3). The predicted travel time on e_3 on path (e_1, e_3) is 1, because the agent starting at time 1 is predicted to enter e_3 on this path at time 12, which is more than 1 time unit later than the previous agent. Thus the predicted arrival time at vertex v_3 on path (e_1, e_3) is 13. The predicted travel time on e_3 on path (e_2, e_3) is 1 too, because the agent starting at time 1 is predicted to enter e_3 at time 11.5 which is more than 1 time unit later than the previous agent. Thus the predicted arrival time at vertex v_3 on path (e_2, e_3) is 12.5. The predicted arrival times at vertex v_3 are 13 on path (e_1, e_3) and 12.5 on path (e_2, e_3). The predicted fastest path is (e_2, e_3), so this agent of the flow selects this path.

The predicted travel time for the agent starting at time 2 on e_1 is 10, because the last agent entered this road more than 1 time unit before. The predicted travel time on e_2 is 11.5, because the agent starting at time 1 entered this road 1 time unit before. Thus the predicted arrival times at vertex v_2 are time 12 on path (e_1, e_3) and time 13.5 on path (e_2, e_3). The predicted travel time on e_3 on path (e_1, e_3) is 2, because the agent starting at time 2 is predicted to enter e_3 on this path at time 12, and one agent entered e_3 within 1 time unit. Thus the predicted arrival time at vertex v_3 on path (e_1, e_3) is 14. The predicted travel time on e_3 on path (e_2, e_3) is 1, because the agent starting at time 2 is predicted to enter e_3 at time 13.5 which is more than 1 time unit later than the previous agent. Thus the predicted arrival time at vertex v_3 on path (e_2, e_3) is 14.5. The predicted fastest path is (e_1, e_3), so this agent of the flow selects this path.

The agent starting at time 2 enters e_3 at time 12, and it does not interfere with the agent starting at time 1, so the actual arrival times of the agent starting at time 1 are the same as its predicted arrival times on path (e_2, e_3).

The predicted travel time for the agent starting at time 3 on e_1 is 11, because the last agent entered this road 1 time unit before. The predicted travel time on e_2 is 10.5, because the agent starting at time 2 entered this road more than 1 time unit before. Thus the predicted arrival times at vertex v_2 are time 14 on path (e_1, e_3) and time 13.5 on path (e_2, e_3). The predicted travel time on e_3 on both paths is 1, because the agent starting at time 3 is predicted to enter e_3 on this path at time 13.5 or later, which is more than 1 time unit later than the previous agent that started at time 1 (and is predicted to enter e_3 at time 12). Thus the predicted arrival time at vertex v_3 on path (e_1, e_3) is 15, and the predicted arrival time at vertex v_3 on path (e_2, e_3) is 14.5. The predicted fastest path is (e_2, e_3), so this agent of the flow selects this path.

The agent starting at time 3 enters e_3 at time 13.5 and does not interfere with the agent starting at time 2, so the actual arrival times of the agent starting at time 2 are the same as its predicted arrival times on path (e_1, e_3).

The rows for the agent starting at time 4 can be filled out with similar reasoning as the rows for the agent starting at time 2.

As Table 5.1 shows, the agent starting at time 1 and the agent starting at time 2 arrive at vertex v_2 and enter edge e_3 at time 11.5 and 12 correspondingly. The agent starting at time 3 and the agent starting at time 4 enter edge e_3 within one time unit as well. This means that, from time to time, two agents enter edge e_3 within one time unit, therefore the flow on e_3 will be 2 for a while, which is more than $r_1 = 1$. $\square$

The above Theorem 5.1 shows that "single flow intensification" may happen in some SNIP online routing games, but it does not happen the same way as in simple naive online routing games. Prediction helps to share the resources, and the flow alternates between the parallel roads e_1 and e_2. Because of this, we cannot prove the same way as it is done in [1] that the worst case benefit of online data in SNIP online routing games may be more than 1. We are giving below an alternative proof in Theorem 5.2 for this.

THEOREM 5.2. *There are simple naive intention propagation online routing games where the worst case benefit of online real-time data is greater than* 1.

Proof. We are going to show how this is possible in the SNIP online routing game $SN_2 = (t, T, G, r, c, k)$ which is described as follows. $T = 10$, modelling one time unit. The network G is given as a four node directed graph with $V = \{v_0, v_1, v_2, v_3\}$ and $E = \{e_1, e_2, e_3, e_4\}$. G and its edges are shown in Fig. 5.2. Note that the edges e_3 and e_4 are parallel, but have different cost functions. The cost function c of the road network is $c_{e1} = 1$, $c_{e2} = 1$, $c_{e3} = 10 + x$, and $c_{e4} = 10.5 + 10 \times x$, where x is the total incoming flow on the edge. The total traffic flow is $r = (r_1, r_2)$ with $r_1 = 1$ on the trip P_1 from source vertex v_0 to target vertex v_3 of G and $r_2 = 1$ on the trip P_2 from source vertex v_1 to target vertex v_3 of G. The trip P_1 has the set of paths $\{p_1, p_2\}$, where $p_1 = (e_1, e_3)$ and $p_2 = (e_1, e_4)$. The trip P_2 has the set of paths $\{p_3, p_4\}$, where $p_3 = (e_2, e_3)$ and $p_4 = (e_2, e_4)$.

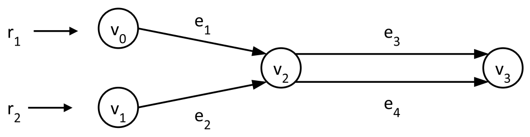


FIG. 5.2. *The network of the SNIP online routing game SN_2.*

In order to compute the benefit of online data, we first determine the non-online cost of the flow r with an oracle using the SNIP decision making strategy but using only the constant part of the cost functions. In this case the predicted cost is the constant part of the cost functions, and the agents always select the shortest path based on the constant part of the cost functions. Thus flow r_1 always goes on the path $p_1 = (e_1, e_3)$, and flow r_2 always goes on the path $p_3 = (e_2, e_3)$. The flow on edge e_3 will be 2, because one agent from r_1 and one agent from r_2 arrive at v_2 in every time unit, and one of them enters e_3 immediately, and the other one with gap_{e_3} delay. We define SN_2 with $gap_{e_3} < T$. So the cost of path $p_1 = (e_1, e_3)$ is $c_{p1oracle} = 1+10+2 = 13$ and the cost of path $p_3 = (e_2, e_3)$ is $c_{p3oracle} = 1 + 10 + 2 = 13$. The total cost of the flow r is $c_{roracle} = 1 \times 13 + 1 \times 13 = 26$ with the non-online oracle.

Now we are going to prove that the cost of the flow r may be above 26 in the SNIP online routing game SN_2. When the flow r starts to flow at time step $t = 0$, the traffic forecaster will predict 10 for the cost of e_3 and 10.5 for the cost of e_4, so both r_1 and r_2 will select the edge e_3. The flows propagate their intentions, so the traffic forecaster will know that a flow of 2 agents will enter edge e_3 from time step $t = 10$ (time step $t = 10$ is time 1, and one agent enters e_3 at time 1, and one agent at time $1 + gap_{e_3}$), and 2 agents will be on e_3 until at least time step $t = 110$ (time 11). Both r_1 and r_2 are 1, so they make their next decision at time step $t = 10$ (time 1). At this time the traffic forecaster tells both r_1 and r_2 that this agent of their flow will arrive at v_2 at the time step $t = 20$ (time 2) facing e_3 with cost 12 (because two agents entered e_3 in the preceding time unit) and e_4 with cost 10.5. Note that the traffic forecaster is only aware of the intention propagations before the current time step, and the traffic forecaster does not forecast the decisions at the current time step. Because the predicted cost of e_4 is smaller than the predicted cost of e_3, both r_1 and r_2 will select e_4. Both will arrive at v_2 at time step $t = 20$ (time 2), and one of them (let's say r_1) will enter the edge e_4 at time 2 with a cost of 10.5, and the other one will enter the edge e_4 at time $2 + gap_{e_4}$ with a cost of $10.5 + 10 \times 1 = 20.5$. We define SN_2 with $gap_{e_4} < T$. So the flow r at the time step $t = 10$ will decide to go on the paths $p_2 = (e_1, e_4)$ and $p_4 = (e_2, e_4)$. The cost of the path p_2 will be $c_{p2online}(1) = 1 + 10.5 = 11.5$, and the cost of the path p_4 will be $c_{p4online}(1) = 1 + 20.5 = 21.5$. The total cost of the flow r will be $c_{ronline}(1) = 1 \times 11.5 + 1 \times 21.5 = 33$ with the SNIP online strategy.

From the above calculations $c_{ronline}(1) = 33 > c_{roracle} = 26$, so the worst case benefit of online data in the SNIP online routing game SN_2 is greater than 1. □

The above theorem points out that worst case benefit of online data below 1 is not guaranteed in SNIP online routing games. There is even stronger statement, because the worst case benefit of online data can be arbitrarily large, making the agents very unhappy, as the next theorem shows. Note, that this is not specific to intention propagation, just this property was not investigated for simple naive online routing games.

Theorem 5.3. *Given any arbitrarily large number α, there are simple naive intention propagation online routing games with cost functions of the form $a \times x + b$, where the worst case benefit of online real-time data is bigger than α.*

Proof. Let us take the SNIP online routing game $SN_2 = (t, T, G, r, c, k)$ from the above Theorem 5.2, and formulate its cost function c_{e4} in a parametrized way by replacing the 10 coefficient with a $26 \times \alpha$ parameter: $c_{e4} = 10.5 + 26 \times \alpha \times x$. Repeating the reasoning as in Theorem 5.2, we get that $c_{p4online}(1) = 1 + 10.5 + 26 \times \alpha \times 1 = 11.5 + 26 \times \alpha$, and $c_{ronline}(1) = 1 \times 11.5 + 1 \times (11.5 + 26 \times \alpha) = 23 + 26 \times \alpha$. The worst case benefit of online data is at least $c_{ronline}(1) \div c_{roracle} = (23 + 26 \times \alpha) \div 26 = 23 \div 26 + \alpha$ which is bigger than α. □

The third problem investigated in [1] is the question of equilibrium, and it is shown that equilibrium is not guaranteed in simple naive online routing games, because they may "fluctuate" at some flow values in some games. Now the question is whether intention propagation can help to avoid "fluctuation"?

Theorem 5.4. *There are simple naive intention propagation online routing games which do not have equilibrium at certain flow values.*

Proof. The SNIP online routing game SN_2 of Theorem 5.2 is an example which does not have equilibrium and shows fluctuation. We are going to continue the scenario of Theorem 5.2, and we show what will be the decisions of the flows after time step $t = 10$. The flow r at time step $t = 10$ decided to go on the paths $p_2 = (e_1, e_4)$ and $p_4 = (e_2, e_4)$, and there will be a flow on the edge e_4 from time step $t = 20$ (time 2) with cost 21.5. Because overtaking is not allowed on the edges, the cost of the edge e_4 will be 21.5 from time step $t = 20$, and this cost will linearly decrease as time goes by, reaching the value 12 at time step $t = 115$ (i.e. 95 time steps later), if no other flow enters the edge e_4. If the full flow r repeatedly goes on e_3, then the cost of e_3 is 12 (see the reasoning in Theorem 5.2), so the traffic forecaster will predict maximum 12 for the cost of e_3 and more than 12 for the cost of e_4 between time steps $t = 20$ and $t = 115$. This means that the flow r will select the edge e_3 until time step $t = 110$. Time step $t = 110$ will be the next decision point when the traffic forecaster will predict value below 12 for the cost of e_4, because it will tell both r_1 and r_2 that the current agent of their flow will arrive at v_2 at time step $t = 120$ (time 12) facing e_3 with cost 12 and e_4 with cost 11.5. Because the predicted cost of e_4 is smaller than the predicted cost of e_3, both r_1 and r_2 will select e_4. Both will arrive at v_2

at time step $t = 120$ (time 12), and one of them (let's say r_1) will enter the edge e_4 with a cost of 11.5, the other one will enter the edge e_4 with a cost of $10.5 + 10 \times 1 = 20.5$ (see the reasoning in Theorem 5.2). So the flow r at the time step $t = 110$ will decide to go on the paths $p_2 = (e_1, e_4)$ and $p_4 = (e_2, e_4)$. The cost of the path p_2 will be $c_{p2online}(11) = 1 + 11.5 = 12.5$, and the cost of the path p_4 will be $c_{p4online}(11) = 1 + 10.5 + 10 \times 1 = 21.5$. The total cost of the flow r will be $c_{ronline}(11) = 1 \times 12.5 + 1 \times 21.5 = 34$ with the SNIP online strategy. After time step $t = 110$, the cycle will start again: the flow r will return to edge e_3, then after a while there will be a short switch to e_4, then e_3 again, etc. The flow r will not have an equilibrium, and its cost will fluctuate between 13 and 34. $\square$

Now we have analysed SNIP online routing games, and we have proved that the three properties proved for simple naive online routing games in [1] are true for SNIP online routing games as well. However we can observe that SNIP online routing games manifest these properties in a different way, which we could describe informally as "not so intensive", because prediction helps to detect the problems even before they actually occur. So intention-propagation-based prediction helps to smooth down the problems, but cannot always prevent them.

6. Conclusions. In this paper we have investigated whether intention-propagation-based prediction solves some problems of collective adaptive systems composed of autonomously self-adapting navigation devices. The formal investigation was done on the SNIP online routing game model, which is a somewhat simplified model of the anticipatory vehicle routing system using delegate multi-agent systems described in [3], because the SNIP online routing game model does not include dynamic re-routing.

Online routing games are a novel formal model of autonomously self-adapting vehicle navigation systems using real-time data. The class of simple naive online routing games are the model of current commercial vehicle navigation systems, because current commercial navigation systems use the simple naive decision strategy, which searches the (possibly multiple objective [2]) shortest path, with taking into account the real-time information available at decision time. It is proved [1] that in simple naive online routing games equilibrium is not guaranteed, "single flow intensification" is possible and the benefit of online data is not guaranteed to be below 1. In practical terms these mean that if the vehicles use simple naive strategy, and they exploit real-time data, then the traffic may sometimes fluctuate: if there is congestion on some road, then many vehicles try to adapt to the situation, they select other route, and they cause congestion on some other road. Congestion may be created by the "single flow intensification" phenomenon as well, because vehicles entering the road network later may select alternative faster routes, and they may catch up with the vehicles already on the road, and this way they cause congestion. The result of all these is that sometimes the traffic may be worse off by exploiting real-time information than without exploiting real-time information. One of the causes of this strange behaviour is that vehicles become aware of congestions with delay, only after the congestion is formed. In order to improve this situation, anticipative adaptation techniques [17], among them the techniques of intention propagation and traffic prediction [3], was proposed.

In this paper we have proved that online routing games with intention-propagation-based prediction may also have "single flow intensification" (although in a somewhat "lighter" way), the worst case benefit of online data may go above 1 (in fact it may be arbitrarily large), and the traffic may fluctuate. In practical terms these mean that if vehicles use simple naive strategy, and they exploit intention-propagation-based prediction, then in some networks and in some cases the traffic may be worse off by exploiting real-time information than without exploiting real-time information.

We have pointed out that one of the causes of this surprising result is that the traffic forecaster predicts future traffic conditions based on the intentions of the vehicles already on the road, but it does not predict the intentions of the vehicles currently making decisions. If several vehicles make decisions at the same time, then they may make the same decision to take the same alternative route to avoid the already predicted congestion, and they cause congestion on the alternative route. Obviously, intention propagation helps the vehicles to detect the possibility of congestion formation before the congestion is actually formed, and thus there is smaller "time window" to make the same "wrong" decision to head towards the newly forming congestion. The technique of intention-propagation-based traffic forecast is therefore an important improvement to the simple naive online strategy.

Although intention-propagation-based traffic forecast is an important improvement, it does not fully take us closer to the goal that was set out in [1]: "we expect that the application of these new strategies will be individually rational choice, and therefore the decision strategies can be implemented in the navigation devices themselves instead of the centralized planning approaches like those of Google Maps and Waze, because some users are reluctant to provide private data for the centralized approach". The implementation of intention propagation and traffic prediction involves a delegate MAS which is similar to GoogleMaps and Waze. Although the delegate MAS is distributed, it plays a kind of "central actor" role, because the delegate MAS may collect data about the agents similarly as Google Maps and Waze.

The results of this paper point out future research directions. One direction is to apply other "social" information sharing patterns [17], like gossiping, among vehicle agents to eliminate the "central actor" role of the delegate MAS, so that the vehicle agents have an integrated view of the intentions of all of the vehicle agents, but none of the agents in the system is able to discover the concrete intention of any other agent. Another research direction is to develop new decision strategies that can help to avoid that many vehicles make the same decision at the same time to cause congestion on the newly selected route. We hope that if such strategy is developed, then it may help us to keep the worst case benefit of online data below 1, and thus we can ensure that the users of navigation systems exploiting real-time data will never experience worse performance with online data than without online data.

Because we based our investigations on the online routing game model, our results can be transferred to other application fields than road traffic navigation. In the road traffic navigation field the goals are trips, the shared resources are roads, and the costs are travel times. For example, in cloud computing the goals are jobs to be done, the resources are computing resources, and costs are execution times; in intelligent manufacturing, the goals are workflows to be executed, the resources are manufacturing phases, and the costs are manufacturing times or real financial costs. These other fields can be modelled with online routing games or with online joint resource utilization games as defined in [1].

Acknowledgments. The author thanks the anonymous reviewers of the previous version of this paper which was presented at the FoCAS workshop of the eighth edition of the IEEE International Conferences on Self-Adaptive and Self-Organizing Systems, and the reviewers of the current version. The comments of the reviewers and the participants of the workshop helped to prepare this improved and extended version.

REFERENCES

[1] László Z. Varga, *Online Routing Games and the Benefit of Online Data*, in Proc. Eighth International Workshop on Agents in Traffic and Transportation, at 13th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-6, 2014, Paris, France, pp. 88-95.

[2] Victor J. Blue, Jeffrey L. Adler, George F. List, *Real-Time Multiple-Objective Path Search for In-Vehicle Route Guidance Systems*, in Transportation Research Record No. 1588, Intelligent Transportation Systems and Artificial Intelligence, vol. 1588, pp. 10-17, 1997

[3] R. Claes, T. Holvoet, and D. Weyns, *A decentralized approach for anticipatory vehicle routing using delegate multi-agent systems*, in IEEE Transactions on Intelligent Transportation Systems, vol. 12, no. 2, pp. 364-373, 2011.

[4] Joachim Wahle, Ana Lúcia C Bazzan, Franziska Klügl, Michael Schreckenberg, *Decision dynamics in a traffic scenario*, Physica A: Statistical Mechanics and its Applications, Vol. 287, Issues 34, pp. 669-681, 2000

[5] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani, *Algorithmic Game Theory*, Cambridge University Press, New York, NY, USA. 2007

[6] *Dynamic Traffic Assignment: A Primer, in Transportation Research Circular Number E-C153*, Transportation Research Board, 500 Fifth Street, NW, Washington, DC 20001, 2011

[7] Tim Roughgarden and Éva Tardos, *How bad is selfish routing?*, J. ACM 49, 2, 236-259 (March 2002)

[8] D. Braess, *Über ein Paradoxon aus der Verkehrsplanung*, Unternehmensforschung 12, 258 (1968)

[9] D.C. Parkes, *Online mechanisms*, in [5] pages 411-439

[10] S. Albers, *Online algorithms*, in Interactive Computation: The New Paradigm edited by D.Q. Goldin, S.A. Smolka and P. Wegner, Springer, pp. 143-164, 2006.

[11] A. Galstyan, S. Kolar, and K. Lerman, *Resource Allocation Games with Changing Resource Capacities*, in Proc. of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2003), Melbourne, Australia, page 145–152, 2003

[12] J. Görmer, J. F. Ehmke, M. Fiosins, D. Schmidt, H. Schumacher, and H. Tchouankem, *Decision support for dynamic city traffic management using vehicular communication*, in Proc. of 1st International Conference on Simulation and Modeling Methodologies, Technologies and Applications. SciTePress Digital Library, pp. 327332, 2011

[13] J. Dallmeyer, R. Schumann, A.D. Lattner, and I.J. Timm, *Dont Go with the Ant Flow: Ant-inspired Traffic Routing in Urban Environments*, in 7th International Workshop on Agents in Traffic and Transportation (ATT 2012) held at AAMAS 2012, pp. 59–68. 2012

[14] Robert Rosen, *Anticipatory Systems: Philosophical, Mathematical, and Methodological Foundations (IFSR International Series on Systems Science and Engineering)*, Pergamon Press, 1985

[15] Tom De Wolf and Tom Holvoet, *Design Patterns for Decentralised Coordination in Self-organising Emergent Systems*, in Engineering Self-Organising Systems, Lecture Notes in Computer Science, Vol. 4335, pp. 28-49, 2007

[16] 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. 2666, 2006

[17] Jose Luis Fernandez-Marquez, Giovanna Di Marzo Serugendo, Sara Montagna, Mirko Viroli, Josep Lluis Arcos, *Description and Composition of Bio-inspired Design Patterns: a Complete Overview*, Natural Computing, Volume 12, Issue 1, pp 43-67, 2013

[18] Sara Montagna, Mirko Viroli, Jose Luis Fernandez-Marquez, Giovanna Di Marzo Serugendo, Franco Zambonelli, *Injecting Self-Organisation into Pervasive Service Ecosystems*, Mobile Networks and Applications, Volume 18, Issue 3, pp. 398-412, 2013

Edited by: Giacomo Cabri and Emma Hart
Received: Dec 9, 2014
Accepted: Jul 7, 2015

DOI 10.12694/scpe.v16i3.1099
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 16, Number 3, pp. 233–247. http://www.scpe.org © 2015 SCPE

APPLICATION OF COLLECTIVE MOVEMENT IN REAL LIFE SCENARIOS: OVERVIEW OF CURRENT FLOCKING SOLUTIONS[*]

BERNÁT WIANDT [†‡] ANDRÁS KŐKUTI [†§] AND VILMOS SIMON [†¶]

Abstract. Today more and more attention is given to collective movement of dynamic nodes, called ”flocking”. The main problem is how to control the nodes to behave as a group and make them move together to destinations (and moreover complete some tasks such as sensing, transporting, etc.). Currently there are many works, which attempt to solve the flocking problem, but very few of them are suitable for real world use. This is because most of the algorithms are overlooking crucial physical limitations or the used model is not detailed enough. In this paper we survey the existing flocking algorithms, including a review of the most important requirements for real application scenarios.

Key words: collective movement, flocking, self-propelled particles, self-organising

AMS subject classifications. 68M, 68U

1. Introduction.

Collective motion, which is one of the manifestations of a more general class of phenomena, called collective behaviour, has attracted many researchers from diverse fields of scientific and engineering disciplines. From the area of biology, the grouping of animals is a natural phenomenon, in which a number of animals are involved in joint movement and formation of a group. Examples [35, 37, 31] are insect swarms, fish schools, herds of wildebeests, etc. The main goal of these works is to figure out a model of the grouping, with respect to the dynamics of animal motion. A more general phenomena of collective behaviour can be observed in atoms and molecules as well, for example, during spontaneous magnetisation [29]. Moreover, the concept of collective action exists not only in animals but also in human societies [15]. The flocking phenomena (the notion of flocking is used as a synonym of collective motion) can be an appropriate solution in many engineering applications as well. Applications of flocking include massive mobile sensing in an environment; parallel and simultaneous transportation of vehicles or delivery of payloads; and performing military missions, such as reconnaissance, surveillance, and combat using a cooperative group of Unmanned Aerial Vehicles (UAVs). A flocking group of robots can perform tasks like exploration of an area [24], autonomous navigation for deployment, surveillance or search and rescue operations. Aircraft flying in formation can use the available fuel supply more efficiently. When the formation is given the flexibility to be able to reconfigure itself, the error tolerance and value of the ensemble increases as it will be capable of performing tasks in more diverse environments. Formation reconfiguration enables the vehicles to mitigate certain errors or faults, such as sensor or actuator failures, failure of a complete aircraft and communication problems.

The first work [40] in this area was a computer graphics animation of a group of birds by Reynolds. His bird-like objects, called ”boids”, moved along trajectories determined by differential equations, taking into account three types of interactions:

Flock Centering Attempt to stay close to nearby boids.
Collision Avoidance Avoid collisions with nearby boids.
Velocity Matching Attempt to match the velocity of nearby boids.

The rules mentioned above are also known as *cohesion*, *separation*, and *alignment* in the literature. These three forces are the core of all the models in the surveyed literature, however, the implementation of them always differs and depends heavily on the used model. Additions to the basic three forces may appear in some works, for example Vicsek et al. in [52] propose an additional fourth force dampening the control inputs and therefore making the flock more stable.

[*]This paper has been partially supported by HSN Lab, Budapest University of Technology and Economics, http://www.hsnlab.hu
[†]Budapest University of Technology and Economics, Department of Networked Systems and Services
[‡]bwiandt@hit.bme.hu
[§]kokuti@hit.bme.hu
[¶]svilmos@hit.bme.hu

Aside from the common rules, there are some common paradigms in all algorithms. All reviewed implementations use the self-propelled particle model. In this model, the states (position, velocity, acceleration) of two individual units change during their interaction. On the other hand, in equilibrium systems the total momentum is preserved and that is how the velocity distribution is being built up. There are three control methods, which are used widely in the literature. Some of the algorithms are *unit-centre referenced*, where only local information is used (information gathered from the node itself), therefore nodes are independent in the ensemble, while others depend on information collected from neighbours, this is referred to as *neighbour-referenced*, but most of them suit the *leader-follower* model, where there is one or more leader(s) of the whole group commanding other nodes.

In contrast to the huge number of flocking algorithms already published in the literature, in our opinion, very few of them are capable of real world use. There are many requirements (such as obstacle avoidance, scalability, tolerance to communication link errors, dealing with noisy measurements and time delays, etc.), that have to be satisfied by an algorithm to succeed in achieving flocking in a real environment. These requirements are usually only partially satisfied by the surveyed algorithms, as for example in a formal proof about the consistency of the flock, the communication errors (such as packet loss) are not taken into account, or in a simulation framework, the real physical conditions cannot be fully simulated.

The rest of the paper is organised as follows: in Sec. 2 we give a short description of our preferences when examining the proposed flocking solutions along with some introduction to the basic ideas involved in the field. In Sec. 3 we give a comprehensive overview of the control schemes involved in flocking systems and in Sec. 4 we categorise the models used in the reviewed articles. In Sec. 5 we review the communication challenges and solutions and in Sec. 6 we describe some of the simulators and demonstrations connected to flocking.

2. The examined properties of flocking solutions. The main focus of our analysis in the following sections revolve around the applicability of published flocking algorithms in real life. We do not have the resources to build the algorithms into real hardware and test them in a variety of environments, therefore we approach this complicated question by categorising the available works based on how elaborate is the used model and trying to find some kind of evolution article by article.

When we talk about flocking and flocking algorithms, the properties of the environment (i.e., the used model) is one of the most important aspects. The restrictions imposed by the environment have effects on the mathematical formulation of the algorithm, the possible communication and movement models, and it defines the strength of the formal proofs given by the authors. There are numerous questions arising when talking about a physical model, such as the boundedness of the control output, the needed communication bandwidth, the available information, the used vehicle model, etc. For example, when trying to avoid collisions, some articles assume an unbounded control output is possible, which is clearly not the case in a real life scenario. Designing precise control for a complicated vehicle involves a much more elaborate physical model, for example one would have to take into account the intrinsic nonlinearities, the external disturbances influencing the vehicle, the varying weight distribution caused by burning the available fuel or by picking up a payload and the inherent noise and imprecisions in the sensors and actuators.

Flocking algorithms usually depend on some kind of information gained through communication. By nature of the nodes involved, the communication is always wireless, which poses another interesting problem. How to model the connection between the nodes in order to be realistic and also try to keep the model as simple as possible? Flocking algorithms usually do not pay attention to communication aspects, such as the wireless technology used, modulation, interference, frequency band, packet loss, wireless link strength, available bandwidth and so on. Most of the models assume a simple graph based communication network, where two nodes with positions q_i and q_j can communicate if $\|q_i - q_j\| < r$, r being the interaction range. The notion of neighbourhood is introduced to represent the nodes in the 2D circle or 3D ball with radius r around the node. This definition of neighbourhood may not be the one that is actually used in bird flocks. In [9] authors conclude, that bird flocks (starlings in this case) are using a topological neighbourhood rule, where the number of neighbours are fixed and selection of neighbours are independent of their physical distance. The simple communication model used in flocking algorithms has a number of different forms, starting with the *undirected*, where we assume that if node i can communicate with node j, then it is true the other way around. In real wireless systems this is not always the case as there can be simultaneous radio transmissions going on around node i and j. *Directed* communication models incorporate this effect, but are harder to reason about. Connectivity, the available communication links and their quality, is usually captured with an adjacency matrix or some kind of extended form of it. Several works discuss the connectedness of the flock and deliver mathematical proofs upon the preservation of connectivity through time. New connections in this model are formed if two nodes are close enough to each other, however this poses a problem, when the distance between them end up around r: the connections between them can form and be destroyed rapidly. This can be remedied by introducing some kind of hysteresis when adding new neighbours [53] and is used by some works that recognised this problem.

Information about the node itself and especially about its neighbours are not always available and also not accurate. If the information needed in an algorithm can be reduced, simpler hardware can be used and fewer assumptions are required in order for the algorithm to fulfill its duty. Some works require the position and velocity information to be available and accurate in order to achieve flocking. Others assume, that the position information is noisy and only the relative distances can be measured accurately. Some depend on a reduced set of available data and define certain tradeoffs, for example, in the second order case (double integrator dynamics) require only the position and velocity information but from the 2-hop neighbours too, or require only the heading angle to achieve flocking but introduce a gain factor in the control equation, which has to be "large enough". In this case this simplifies the whole system, but introduces larger oscillations between nodes due to the large control gains. These oscillations can result in a quite erratic behaviour of the flock and eventually can cause the flock to disperse. In [52] the authors recognised this problem and proposed a damping factor in the control output to prevent such oscillations. They use the same approach in the implemented control algorithm on their drones [51], a real world, truly autonomous flocking implementation of UAVs.

Simulations are usually done to prove the claimed properties in one or more trials, but simulators can contain bugs, or simulations can be set up to show the best case scenario for a proposed method. Also simulators suffer from the same problems as described in the beginning of this section, namely the simplistic assumptions about the real world, mobility models and communication. Therefore the ultimate proof of a mathematical formula or a proposed algorithm is the real world trial, where the authors take the effort of implementing the solution to actual physical nodes and run, measure and evaluate performance characteristics of their work. We do know that all of these requirements is impossible to meet in one publication but we strive for these and will try to show the reader a way among the vast amount of work done in this field up to date.

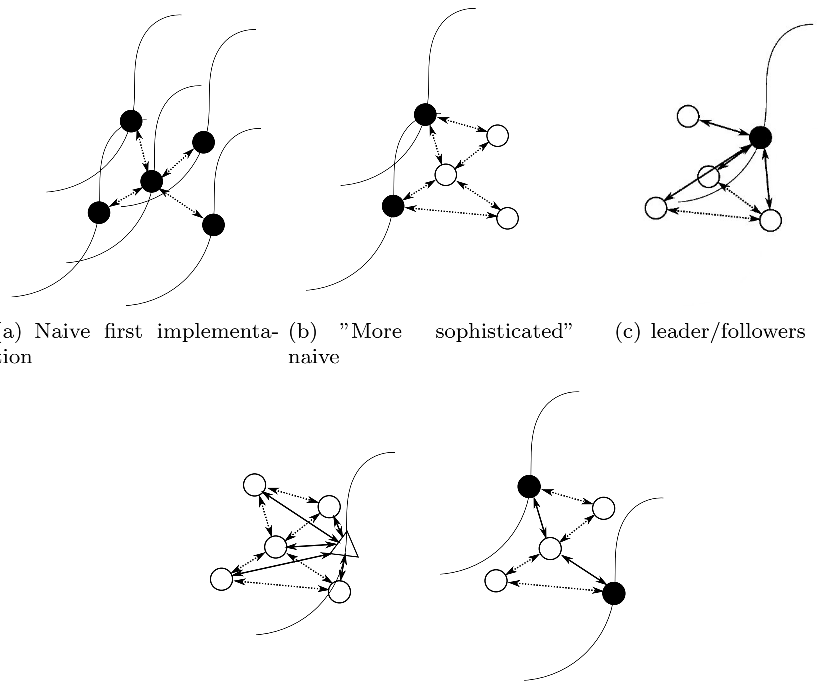
3. Control mechanisms for flocking. A very important property of a flocking algorithm is the control mechanism, which is responsible for directing the flock to the desired destination and managing the shape of the group. Various mechanisms have been developed for different use cases to be introduced in this section.

The simplest implementation is if each node knows the trajectory or the previously determined direction [40], in which case no control overhead is needed. However, in a real life scenario the trajectories or even the directions are not previously determined and sometimes both parameters are changing from time to time (in a real flock it is essential to change the destination on the fly, for example in a search and rescue operation).

A more sophisticated mechanism is proposed in [13], where the nodes can be separated into two distinct sets. In the first one each individual is "naive", which means it does not possess any kind of trajectory or directional information, thus nodes in the first set can only follow the flock. The nodes in the second set are "informed", hence they know the desired trajectory or direction. In this model, the second set of nodes do not indicate that they are "informed", but instead guide the rest of the swarm by moving in the preferred direction. With this mechanism the user manages only a few of the interacting nodes, but due to the used flocking model, they can influence each participant in the swarm. Findings and results from several references show that a self-organised flock of nodes can be effectively guided by a minority of informed nodes within the flock. However, a huge shortcoming of this solution is that the swarm will only slowly converge to the preferred direction, since the "informed" nodes are not turning instantly, instead the whole flock will turn to some weighted sum of the current and preferred direction, and in the next time slot they will turn more closely to the desired direction and so forth, which adds a rather large delay to a target tracking application.

To overcome the previously described problem, the *leader/followers* [33] solution can be employed, which is a widely used control mechanism in the literature [34, 1]. In this control scheme a special role is assigned to a node (called the "leader") to guide the whole flock (and perhaps to maintain the formation). The main difference from the two distinct set scheme is that other individuals (called "followers") are following the leader, thus they know the existence of a leader in the flock who should be followed. A serious disadvantage of this type of control mechanism is that it does not scale well with an increasing number of nodes since the position error propagates cumulatively by disseminating the information from the leader to followers, and that all nodes have to communicate (directly or indirectly) with the leader, while the wireless radio ranges are usually bounded.

The enhanced version of the previous solution is the *virtual-leader* [25] or *virtual-leader/followers* [36] mechanism. Contrary to the *leader-followers* scheme, where the leader is one of the physical agents (e.g., neighbouring a vehicle in a multi-vehicle system or a fish in a school), here it is only a virtual reference point (beacon), that can influence the neighbouring nodes. In this model a new force is introduced, guiding the nodes, similar to the inter-node forces, except that it defines the force on a physical node in reference to a virtual leader. Since there is no physical node with a special role (such as the leader in the previous scheme), all of the nodes are interchangeable with each other. The main benefit of this approach is the robustness of the group to the failure of an individual agent. Because of its beneficial properties, it can be found in many flocking solutions [56, 19, 49, 53].


(a) Naive first implementation (b) "More sophisticated" naive (c) leader/followers

(d) virtual-leaders (e) pseudo-leaders

Fig. 3.1. *A simple example for each of the described mechanisms. The agents are represented with circles (the filled ones possess information about the desired trajectory / direction) the triangle indicates a virtual agent (or a reference beacon), while the arrows signal the inter-node forces (potentials). The dashed arrows present a usual interaction (such as cohesion, separation, etc.) while the solid ones indicate the controlling interactions. Controlling interaction is used only in cases when a follower node is informed about a leader to be followed.*

The extension of the *virtual-leader/follower* mechanism is the *pseudo-leaders* [65] model. This model is very similar to the previous one except that it is not essential for all agents to be informed [43], and a pseudo-leader represents an agent, while in the previous model there were just reference beacons. A flaw of this solution is that the selection of pseudo leaders [65] works only on a fixed topology, while in a real life scenario, in the majority of the cases, the topology varies dynamically.

There are other schemes [45, 4], where no leaders are designated and nodes react to the positions of their neighboring nodes, while moving in an emergent direction, but their impact to the flocking algorithms is less compared to the previous models.

For a better understanding, the described mechanisms are depicted in Fig. 3.1. This figure does not show the whole picture, since not all inter-node interactions (depicted with arrows) are illustrated. The controlling mechanisms differ from each other in how many leaders exist in the flock, how they control other nodes, and how the other nodes are informed via communication links. As it can be concluded from this section, constructing an appropriate controlling mechanism to be feasible in real life is a huge and complex task, therefore some widely used patterns are the basis of almost every flocking algorithm.

4. Physical models for flocking algorithms. The physical model assumed in any work constrains the control algorithm, influences the capabilities of the system and is one of the most important factors when designing the control algorithm. The model describes the dynamics of the flock members, the interactions between them, possible environmental noises and external disturbances. Let us enumerate the different physical models involved in the reviewed research, starting from the simplest to more detailed and complicated. The models we examined are based on the assumption, that the vehicle or flock member is particle-like and it can propel itself in the direction and with the velocity desired, based on the used control algorithm. Usually the size of the vehicle is not considered in the models, rather it is accounted for when designing the collision avoidance part of the control algorithm, by enforcing a constraint on the minimal distance between two vehicles. The simplest particle-like model is a linear-integrator type system, the simple integrator, also known as first-order kinematics, defined by Eq. 4.1.

$$\dot{q_i} = u_i, \qquad i = 1...n \text{ and } q_i, u_i \in R^m \text{ (e.g., } m = 2, 3) \tag{4.1}$$

In first order kinematics, the control input is the velocity vector of the vehicle. This is the easiest model, it is far removed from the real world but is easy to reason about. It is used as a preliminary analysis tool in [20, 60, 57]. A far more usable and widespread model is the double integrator, a.k.a. second-order kinematics. The model is defined by Eq. 4.2.

$$\dot{q_i} = p_i \text{ and } \dot{p_i} = u_i \qquad q_i, p_i, u_i \in R^m \text{ (e.g., } m = 2, 3) \tag{4.2}$$

being the position and velocity of the node i respectively, and u_i is the control input for node i. A basic assumption is that the node can be controlled in any direction, it does simple integration, it has no weight or other interfering forces from the outside world. The control algorithm is designed to generate u_i, based on the input variables, such as the current position, velocity and acceleration of node i and some subset of the same information from its neighbours. Information flow between nodes is crucial, as control algorithms need input data to generate the correct output, therefore move the node in the desired direction. One of the used descriptions of communication is the unit-disc model, where nodes within a specified distance in space can communicate with each other. These models are usually undirected, meaning that for all participants: if node i can communicate with node j, then node j can communicate with node i. This model is prevalent and used for example in the landmark paper of Olfati-Saber [36], where they study the possibilities of flocking with a virtual leader-follower type model. The solution for the flocking problem in [36] is based on virtual potential functions, such as the three basic forces (cohesion, separation, alignment) is modelled with potential functions either attracting or repelling other nodes.

The Olfati-Saber approach to flocking assumes that information about the virtual leader is available to all flock members, which is clearly not practical, nor scalable. Subsequent works in the field proved that flocking can be achieved, when only a (small) fraction of the nodes are informed about the virtual leader's state [47, 48, 43]. It is a beneficial property, because this way the flock needs less communication and the uninformed members can be driven by the informed ones. Although in [43] it is proven, that a subset of nodes is enough to be notified of the virtual leader's state, the selection algorithm was not provided and a clear rule as to which nodes should be informed was not given. In [21], authors give a criterion to choose the nodes to be pseudo leaders, therefore they provide a tool to actually implement the previous approach. Pseudo leaders in this context mean the nodes informed about the virtual leader's state. Virtual leaders can be static or dynamic: in the static case their velocity vector is static and in the dynamic case it can be time-varying. Dynamic virtual leaders opened the possibility to target tracking applications with flocks and enabled the free movement of the flock in space.

Dynamic virtual leaders were investigated in [48], which is an extension to the classic Olfati-Saber approach, and is an assumption for almost all reviewed works.

The information required to achieve flocking is limited in [10], as in the first-order case the velocity measurements are not required and in the second-order case accurate acceleration measurements of the leader and followers are not required. Reducing input to the control algorithm is a desirable property, because accurately measuring acceleration and velocity and transmitting it to the neighbours in a timely fashion is not easy to implement, if at all possible, without considerable time-delays and/or inaccuracies. There could be hardware or budget limitations, which make it impossible to use that information. Limiting the input variables of the flocking algorithm makes it easier to implement it and makes it more robust, because higher order information (velocity compared to acceleration and position compared to velocity) can be measured more accurately and with simpler hardware. There has been a lot of effort dedicated to this subproblem, for example [61] for flocking, using only sampled position data or [66] for attitude coordination with the same constraints or [64], which deals with the case, where the flock consists of heterogeneous agents, meaning their movement dynamics and constraints are not the same.

Physical implementation of the controllers on each node has some limitations. Usually this manifests in the boundedness of the control input, that can be satisfied by the controller. This means that we should take this effect into account when developing the control algorithms to achieve flocking using physical hardware. Two of the earliest works dealing with this subproblem are [28], researching flocking in general, and [22], which details an algorithm for flocking with a virtual leader with time-varying velocity. There was some research conducted in the direction of implementing velocity-free bounded controllers, but early works were concentrating only on the velocity consensus or were assuming a fixed interaction topology [64, 2, 3]. Recently, in [16], authors introduced a velocity-free and bounded controller design.

Another interesting subproblem is obstacle avoidance, where the problem is twofold: on the one hand we have to ensure that the nodes will not crash into each other, on the other hand we somehow have to avoid crashing into some bigger obstacle, like a wall. The first half of the problem (usually referred to as collision avoidance) is usually solved by introducing some requirement on the minimal distance between nodes, and ensuring that they will be met at all times by factoring this requirement into the designed virtual potential functions, where the dimensions of the used vehicles can be accounted for. The second half of the problem, avoiding collision with bigger, static or dynamic obstacles was addressed in [36], where the obstacles were modelled as a virtual node in the system. In that work, artificial potential functions (APFs) were used to calculate the control output and the obstacle nodes were designed to allow the nodes closing by to move past them in an efficient way. An interesting approach was proposed in [54], where the authors argue, that APFs have the problem of local extrema (multiple obstacles in an unfortunate configuration, stuck nodes, etc.) and moving around an obstacle will sometimes fail. In their work, stream functions were used to calculate the paths of nodes around obstacles, ensuring stable flocking and connectivity maintenance during obstacle avoidance.

A straightforward extension of the second-order model is to incorporate the intrinsic nonlinearity of the vehicle into the model and move a bit away from the ideal world. The equations for the general case are defined in Eq. 4.3.

$$\dot{q_i} = p_i \text{ and } \dot{p_i} = f(p_i, q_i) + u_i, \text{ where } q_i, p_i, u_i \in R^m \text{ (e.g., } m = 2, 3) \tag{4.3}$$

$f(p_i, q_i)$ represents the intrinsic nonlinear characteristics of the nodes. Recently, nonlinear models got a lot of attention, as they are one step closer to reality. In earlier works dealing with consensus problems (matching velocities among flock members), the authors studied first- and second-order systems with intrinsic nonlinear characteristics, using pseudo-leader [65] or virtual leader [19] approaches. In [65, 19] the nonlinear term in the model is only dependent on the position of the node, however a more realistic model would also depend on the velocities of the nodes. Also, a globally Lipschitz-like condition is generally used in order to formulate appropriate control, and prove that all the agents can synchronise with the leader(s) in [65, 19, 58] and in the second order case in [59, 46]. Based on those two observations, Su et al. proposed an adaptive control algorithm in [49] for flocks with virtual leaders, where the nonlinearity is only locally Lipschitz and the nonlinearity terms in the model depend both on the velocity and position of the nodes, therefore the model used in their article is more general than the previous approaches.

Previously we stated, that second-order systems are the most prevalent and in [10] it was proved, that for simple second order systems there is no need to have all the available information about the neighbours and virtual leader(s) to achieve flocking. In [53] the authors argue, that nodes might be governed by nonlinear dynamics and nonlinear dynamics is commonly used in synchronisation of complex dynamical networks [55, 26, 58]. In [18] the model is a bit more evolved as measurement uncertainties and external disturbances are accounted for and a neural network approach is proposed to achieve consensus with first- and second-order dynamics. This algorithm is dealing only with achieving consensus and rendezvous in a prescribed place. The work carried out in [14] extends this by solving the leader-follower problem, allowing the leader to have a time-varying state trajectory. Up to this point we only talked about agents with uniform intrinsic dynamics, but in real world applications heterogeneous dynamics should be accounted for. Wang et al. proposed an approach for this exact situation in [53], allowing for heterogeneous nonlinear dynamics for the leader and the followers, implementing collision avoidance by defining APFs and introducing a hysteresis-like method for acquiring new neighbours.

Another model proposed, is the Standard Vicsek Model for self-propelling particles, which in it's simplest form is a cellular automaton like approach to describe the motion of self-propelled particles. In this model each node updates its velocity vector by averaging their neighbours' heading angles and adding some random perturbations, to model the various factors influencing the motion of the nodes.

$$v_i(t+1) = v_0 \frac{\langle v_j(t) \rangle_R}{|\langle v_j(t) \rangle_R|} + perturbation \tag{4.4}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{4.5}$$

Equations 4.4 and 4.5 define the model, where $\langle ... \rangle_R$ denotes the average of the nodes' velocities in a circle of radius R. Perturbations in the simplest case are random angles added to the average calculated in $v_i(t+1)$. In this model if some nodes decide to turn, their turning information will be damped by their neighbours' averaging of the heading angles and this will result in a slow and uncoordinated turning motion. In [12], authors recognised that in order to reconstruct real flocking motion while turning, there needed to be a conservation of rotational symmetry implemented in the system and there needs to be a behavioural inertia mediating the effect of the social force. Therefore they propose a new model, from which the Standard Vicsek Model can be obtained as the over-damped limit of the system. Members of the flock derive their velocity vector from their neighbours and the selection of neighbours can have a huge effect on the stability and structure of the flock. In the simplest case, neighbours are the nodes around us and closer than a predetermined distance, usually called the interaction range. In 2D this defines a circle, in 3D a ball around each node. However, real flocks (for example bird flocks) do not function that way. They select the neighbours from which they derive information in a topological manner. In [9] authors test the hypothesis, that proper topological neighbour selection is beneficial, and conclude that compared to the metrical neighbour selection scheme (circle or ball around the nodes), it resulted in a more stable and more cohesive structure. Topological interaction means that each node interacts with a fixed number of other nodes, irrespective of their distance. Implementing this with real hardware of course has its limitations, because of the limited range of wireless communication.

The models we investigated are trying to resemble the real world, with increasing complexity, external disturbances and nonlinear intrinsic dynamics. A common shortcoming of these algorithms, mentioned in [23], is that the vehicles in the flock or formation may have additional constraints. For example, the more and more popular UAVs involved in formation flight cannot endure negative velocities if they are of the fixed wing type, therefore the control should be designed with this constraint in mind. A particular subproblem investigated in the article is turning: if the leader of the formation turns too fast, some vehicles should decelerate in order to maintain their position and this consequently limits the turning angle of the leader.

Authors in [39] demonstrated the benefits of using multiple UAVs in a target tracking scenario. In their application, the target identification was implemented by image recognition and this resulted in more constraints on the model, for example the turning speed of each vehicle was maximised, because fast enough turning would be detrimental in the performance of the image recognition algorithm. Their physical model is highly detailed, incorporates the gravitational force, environmental disturbances (i.e., wind), and is chosen to be a 4 degree of freedom stochastic model. The stochasticity in the system should capture forces missing from the model and environmental disturbances as well.

In Table 4.1 we summarised the properties of some of the solutions mentioned in this section. In the first column there is a symbolic name, usually the last name of the first author and the reference number. The second column titled **Control** is the type of control scheme that was applied in the algorithm. Empty cells mean that there is no explicit control, and the problem solved is a consensus problem, meaning that the goal is to synchronise the velocity vectors of the nodes in the flock. *Virtual leader* means that there is a (virtual) node to be followed by the others in the flock. *Constant* means constant velocity and *dynamic* means variable velocity for the virtual leader. The third column is titled **Informed** and it contains the number of nodes that has to be informed of the leader's state. Consensus problems do not have a leader, so the cell's value is *N/A*, other solutions contain either *all* if all the nodes have to be informed or > 1 if informing at least one node is enough for the flock to achieve its goal. The fourth column is **Linear**, *yes* if the solution assumes linear dynamics for the nodes and *no* if the nodes have intrinsic nonlinear characteristics. **Noise** column is *no* if the algorithm disregards internal uncertainties and external disturbances, *yes* otherwise. **Input** enumerates the needed information about the neighbours in the system. In two cases ([18] and [14]) there is no exact input variable mentioned in the article, just a general state vector is exchanged possibly containing the position, velocity, temperature, etc. Seventh column is called **Heterogeneity** and it distinguishes whether the algorithm considers homogeneous nodes or different dynamics and properties are allowed for the nodes that make up the flock. *No* signifies the homogeneous case, while *yes* means a heterogeneous flock. Collision avoidance is always implemented in the reviewed works, while obstacle avoidance is not. The column titled **Obstacle avoidance** contains a *yes* if there is some sort of mechanism helping the flock overcome arbitrary sized and shaped obstacles in the space, and *no* means that the authors assumed free flocking, meaning there are no obstacles in the field. Lastly **Communication** gives an overview of the properties of the communication assumptions. *varying* stands for a variable neighbour set, *fixed* for a fixed one. *unit-disc* means a unit-disc model, while *undirected* or *directed* distinguishes between the connection types in the communication graph describing the system.

5. Communication between the nodes.

Research papers in this field fall mainly into two categories. The first group is composed of solutions that rely on the existence of sensors dedicated to measure the distances between nodes, such as infra-red and/or ultra-sonic sensors [34, 1, 42] or the infra-red short range sensing systems [13]. The second group of solutions are significantly different from the first, since nodes are only measuring their own location (via a GPS receiver) and periodically broadcasting this location information via their radio interface. Solutions from the first approach do not depend on the usage of the radio, however, they work only when nodes are relatively close to each other. Some of these solutions are already used in use cases when smaller areas need to be covered with less mobile nodes, e.g., robots on the ground. The second group is targeting highly mobile and dynamically reshaping flock of nodes, e.g., UAVs, especially for the scenarios when the nodes can autonomously split and rejoin. Algorithms from the second group must solve the problems arising from wireless communication, such as medium usage, asynchrony, packet losses, etc. In this section the communication models and methods from the second category will be described, currently in use by well-known flocking algorithms.

The simplest model is an all-to-all broadcast, meaning that any node can communicate with any other node. In this ideal case the radio range is infinite, the communication is undirected and there are no packet losses assumed. When having such assumptions, there is no need for a sophisticated broadcast method, therefore blind flooding [27] can be applied. Of course this model does not meet any requirement of a real application, but it is used widely [12] for an initial model, since the first priority of the algorithms is to work under ideal conditions.

A bit more realistic [63, 36] case employs a finite radio range. This is an initial model as well and the use of a sophisticated communication protocol is pointless as well, however the communication between nodes is bounded with a finite range (defining a circle in 2D and a ball in 3D around the node). Due to this limitation, the flocking algorithm is more complex in the case of a leader-follower control scheme, as the interaction range is limited, but the current state of the leader has to be forwarded to every node in the flock. This can cause all

Table 4.1
Summary of a few selected solutions

Solution	**Control**	**Informed**	**Linear**	**Noise**	**Input**	**Heterogen nodes**	**Obstacle avoidance**	**Communication**
Olfati-Saber's [36]	Virtual leader (constant)	all	yes	no	position, velocity	no	yes	varying, unit-disc, undirected
Su's [47]	Virtual leader (constant)	> 1	yes	no	position, velocity	no	yes	varying, unit-disc, undirected
Su's [48]	Virtual leader (dynamic)	all	yes	no	position, velocity	no	yes	varying, unit-disc, undirected
Shi's [44]	Virtual leader (dynamic)	> 1	yes	yes	position, velocity	no	no	varying, undirected
Jin's [21]	Virtual leader (dynamic)	> 1	no	no	position, velocity	no	no	varying, undirected
Cao's [10]	Virtual leader (dynamic)	> 1	yes	no	position, velocity	no	no	varying, undirected
Yu's [61]	-	N/A	yes	no	position	no	no	varying, directed
Zou's [66]	Virtual leader (dynamic)	> 1	no	no	position	yes	no	varying, undirected
Housheng's [19]	Virtual leader (dynamic)	> 1	no	no	position, velocity	no	no	varying, unit-disc, undirected
Yu's [58]	Virtual leader (dynamic)	> 1	no	no	position, velocity	no	no	varying, unit-disc, undirected
Yu's [59]	-	N/A	no	no	position, velocity	no	no	varying, directed
Su's [46]	Virtual leader (dynamic)	> 1	no	no	position, velocity	no	no	varying, unit-disc, undirected
Su's [49]	Virtual leader (dynamic)	> 1	no	no	position, velocity	no	no	varying, unit-disc, undirected
Wang's [53]	Virtual leader (dynamic)	> 1	no	no	position, velocity	yes	no	varying, unit-disc, undirected
Hou's [18]	-	N/A	no	yes	state vector	yes	no	fixed, undirected
Cheng's [14]	Virtual leader (dynamic)	> 1	no	yes	state vector	yes	no	fixed, directed
Wang's [54]	Fixed point	N/A	yes	no	position, velocity	no	no	varying, unit-disc, undirected

kinds of problems, for example increasing error in the perceived position of the leader as we move farther away from it, since there could be nodes, which are not connected directly to the leader and information has to be propagated to them in a hop by hop manner, resulting in large latencies, therefore inaccurate information.

In the previous models the communication graph (where vertices are nodes and edges indicate the communication links among them) was undirected, moreover, in the simplest model the whole graph was complete,

Table 5.1
Comparison of the network model being used in flocking systems

Solution	**Active or passive**	**Transferred data**	**Transmission range**
Shi's [42]	passive	N/A	N/A
Navarro's [34, 1]	passive, active	velocity and weights	3.3m
Çelikkanat's [13]	passive, active	heading vector	$\sim$ 20m
Cavagna's [12]	active	position	∞
Hai-Tao's [63]	active	position	r, a distance limit
Olfati-Saber's [36]	active	position	r, a distance limit
Zhang's [62]	active	position, velocity	r, a distance limit
Carlési's [11]	active	velocity and control variable	∞

and in the second case it is composed of many smaller (the size is determined by the interaction range of the device) complete graphs. A realistic communication environment cannot be described as an undirected graph, instead some works have adopted a directed graph, since the transmission ranges can differ from node to node. It is a valid extension, based on the assumptions, that nodes of the flock could be different devices (such as in [64, 62]) or they can vary the power of their wireless transmitter for energy efficiency reasons or the different number of ongoing transmissions can cause varying interference around them. However, neither the medium access, nor the packet collisions are taken into account in this model, hence the naive blind flooding protocol can be used in this case as well.

In a more realistic scenario, the blind flood protocol cannot be efficient anymore, since the radio channels (for instance in an urban environment) can be heavily loaded and the periodically initiated broadcasts can cause many collisions and even broadcast storms. In [11] communication is supposed to be limited, possibly subject to random failures, and asynchronous. Two different scenarios have been identified: a low and a high rate scenario. In the *high communication rate* scenario it is assumed, that the number of exchanged messages can be significantly larger than the updating command rates of the nodes, while in the *low communication rate* scenario this relation is the opposite. The command rate in this case is the frequency of the control algorithm, meaning that new control output is generated with the command rate. Nodes have control variables, used when calculating the control output for each node. These variables can be altered by the nodes and periodic updates are needed to accurately approximate their neighbours' states (position, velocity, etc.). Based on these assumptions, an adaptive algorithm was proposed: in the *high communication rate* phase it switches to a vanishing step-size gradient based distributed algorithm to approximate the appropriate control output, where the agent updates its control variables according to a local gradient descent. At the end of the updating phase some randomly selected agents transmit their control variables to their neighbours. In the *low communication rate* phase it is no longer possible to exactly determine the control variables (since messages cannot be transferred fast enough), therefore, a simulation based approximation is employed. Random gossip (namely Pairwise Gossip [8] and Broadcast Gossip [5]) models were used in the paper, since they allow to accurately model the randomness of the links between agents and do not rely on any possibly constraining scheduling of the communication between agents.

Table 5.1 summarises the solutions discussed in this section. The communication between the nodes can be passive and/or active. Passive means that there is no data transferred via wireless interface, thus the system depends only on data gathered from the sensors (such as infra-red sensors or ultra-sonic sensors). While in the active case the data is grouped into messages and disseminated over the wireless network formed by the nodes. In solutions where only passive communication is used no data is transferred and no transmission range defined. There are some cases however, where the passive system is used, together with a communication network. In those cases, the control algorithm needs additional data disseminated over the communication network. In the last column the transmission range is described for each solution.

In addition to the table it can be concluded that all the examined solutions (except Carlési's [11]) use an infinite communication medium. Therefore, there is no need for a medium access protocol since there is no packet losses and each node can transmit messages at any time regardless of the medium usage. This assumption is essential to exchange information (such as position and velocity) with a very high frequency (*e.g.* 10 times in a second) and with the periodically received messages the control model can calculate different type of parameters from the received data (such as velocity form the position or acceleration from the velocity) with relatively high accuracy. Only the framework proposed by Carlési et al. [11] triestime slot to tackle the problems of communication failures, but still many idealistic assumptions are used for the communication environment, such as the radio range being infinite. The scalability of the network is usually not examined: when the flock gets bigger (consisting of more and more nodes), the number of the transmitted messages in a single timeslot increases as well, but since the communicational environment is almost perfect, the messages will not collide. The almost perfect communicational environment is assumed in some of the algorithms [11, 51], however an error is added to the received data, but it is utilised at control layer and not at the communication layer.

To summarise the section, it can be concluded that the current solutions cannot be used in real life use cases, since the communication environments being used do not meet the real life requirements. This is one of the main reasons why nearly all of these solutions work well only in simulations. For example, the algorithm proposed in [51] can achieve flocking only in a rural area, where the external radio noises are much lower than in an urban environment.

6. Closer to real life: simulation and demonstration of flocking systems. As it can be seen from the previous sections, there is an abundance of theoretical models and calculations to describe the properties of collective systems. But what happens when applying such models to real use cases? When examined, only a smaller fraction of the works from the literature introduce simulation measurements and experiments to study the stability and scalability under realistic environmental conditions (for example in the presence of unpredictable environmental noises), the majority of the solutions are not tested in real use cases, remaining only a mathematical basis for further research. The papers with real use cases conduct a systematic set of experiments using both physical and simulated robots, vehicles and UAVs, analysing the characteristics of the flocking solutions and checking if the results are in accordance with the ones that were predicted by the theoretical models. In this section we would like to provide an overview of the existing simulators and robots used to prove the applicability of the previously described algorithms.

Many of the simulators found in the literature are more or less applicable as a real-life simulator. In [34, 1], a dynamic model (the Webots simulator [30]) of the Khepera III robot was used. In the simulation the robots are equipped with distance sensors (infra-red and ultra-sonic), thus they can read the relative positions of their neighbours. The estimation is performed by a relative positioning system, based on the strength of the infrared signals interchanged by neighbouring robots. Robots are able to communicate via short messages, exchanging a unique id, necessary for the proposed algorithms of the system. They can also communicate periodically using the IEEE 802.11 wireless standard and UDP messages. Both communication systems make up two parallel networks, whose links are limited by the distance between robots in order to keep communications local. Messages are always sent from one robot to one of its communication neighbours. The framework allows robots to move in groups, harmonising the group velocity and avoiding obstacles by changing the direction of the group and the inter-robot distance. This is achieved by the robots by agreeing on a reference orientation without the use of any external reference, compass or global positioning system. As a result, robots share a distributed virtual compass, agreeing on a reference orientation. The framework is also equipped with a module enabling the group of robots to split, giving the decision on when to split and in which manner arbitrarily by one of the robots. The introduced framework can be implemented on any mobile robot capable of calculating the relative positions of adjacent robots and communicating with them. The simulator used for the simulated experiments was Webots, using a dynamic model of the Khepera III robot. The Webots simulator is a very realistic simulator, since the implementations can be transferred to real robots directly, but its disadvantage is that the robots can only move in a 2D plane because of the Khepera III specification. Up to 40 Khepera III robots were used in simulation, demonstrating experimentally the scalability with an increasing number of robots. The authors did not stop at the simulation level, they have further developed and tested a framework for the collective movement of mobile robots. The sets of experiments with real robots were as similar as possible to those in the simulation in order to compare them.They have performed experiments with real robots (using eight from the above mentioned Khepera III robots), proving that the framework works properly in a realistic environment. The join/split capabilities of the framework have been successfully tested both in simulation and using real robots. The framework cannot deal with movement in three dimensions, however the authors claim that by modifying several modules it could be accomplished.

Another robot based simulator was presented in [13]. This physics-based simulator, called Controllable-Swarm Simulator (CoSS) [50], contains virtual Kobot based robots, which have two differentially driven motors and infrared (IR) sensors (the infrared short-range sensing system, which is composed of 8 IR sensors placed 45° intervals) around the base. Crosstalk among the nearby robots is avoided using the CSMA-CA (carrier sense multiple access collision avoidance) algorithm during sensing. For the wireless communication an IEEE 802.15.4/ZigBee module is used, with a range of around 20 m indoors. A virtual heading sensor (VHS) is utilised, composed of a digital compass and a wireless communication module, allowing the robots to sense the approximate relative orientations of nearby robots. The heading information is broadcasted by the robots to other robots within their communication range, converting the headings of the neighbouring robots to a local reference frame of the robot by vectorial subtraction of the robots own heading, for aligning the robot with its neighbours. The CoSS was used for performing experiments with more robots than physically available and for longer durations than possible in the physical experimental setups. It is using the ODE (Open Dynamics Engine) physics engine to study flocking behaviour with a large group of robots. Beside the simulated robots, they have carried out a wide range of experiments utilising physical robots, analysing the transient and steady-state characteristics of steered flocking, showing that the results conform with the theoretical models introduced earlier. The swarm of these mobile robots, connected via proximal sensing, were able to wander in an environment by moving as a coherent group in open space and avoiding obstacles.

The ARGoS simulator [38] is a bit different from the previously described solutions. Its main design focus is pointed at the simulation of large heterogeneous swarms of robots. Heterogeneity means in this case that there are different types of robots, which are responsible for special tasks e.g. a hand-bot [7] is used to climb a shelf and take something, while the eye-bots [41] are designed for coordination. The main advantage of this simulator is the open source implementation, together with the possibility to utilise various types of robots for the given use cases. The latter is achieved by dividing the simulated space into non-overlapping sub-spaces, each controlled by a separate physics engine. The rules implemented in each physics engine can be tailored to optimise the run-time of an experiment. Sensors and actuators are plug-ins that access the state of the simulated 3D space. Sensors are granted read-only access to the simulated 3D space, while actuators are allowed to modify it. ARGoS provides an abstract control interface that controllers must use to access sensors and actuators. The same control interface is also implemented on the real robots. In this way, the user code developed in simulation can be transferred to the real robots without modifications. The multi-threaded architecture of ARGoS can provide an astonishing performance of accurate 2D dynamics simulations of 10,000 robots in 60% of real time, furthermore accurate 3D dynamics simulations with the same number of robots in about real time. Because of its properties, it is widely used such as in [17, 6, 41, 7, 32].

A novel realistic simulation framework was introduced in [52] and used in [51]. It was developed to test and optimise two of their algorithms of collective motion: a self-propelled, bio-inspired flocking algorithm and a target tracking setup before employing them in a real scenario. The framework takes into account many of the inaccuracies of a real system, such as the position and velocity measurement error, furthermore the general Gaussian noise corresponding to the environmental effects, the low update rate of the GPS, the limited range of the communication, etc. The stability of the two algorithms was analysed thoroughly and it was discovered that the instabilities can be reduced with optimal strength of the viscous friction-like term. The optimised algorithms were tested on a group of real autonomous robots (quadcopters with on-board computer, GPS device and XBee communication module). They have presented a decentralised multi-copter flock that performs stable autonomous outdoor flight with up to 10 flying agents without central data processing or control.

For flocking algorithms to be applicable in real life use cases, experiments performed on real devices are essential. Successful experiments represent a direct proof of the applicability of the model and the algorithms and also show the stability of the algorithms when the system is exposed to unpredictable environmental noises.

7. Conclusion. Despite of the large number of flocking algorithms already presented in the literature, only a few of them are capable for real world use cases, since there are many requirements that have to be satisfied by an algorithm to enable flocking in a real world environment. This was the reason that the main focus of our survey revolves around the applicability of the published flocking algorithms in real life. We have tried to approach this complex question by categorising the available works based on how elaborate the used model is and trying to find some kind of evolution article by article.

First we have also presented a comprehensive overview of the control schemes involved in flocking systems, which are responsible for directing the flock to the desired destination and managing the shape of the group. Constructing an appropriate control mechanism to be feasible in real life is a huge and complex task, therefore, there are some widely used patterns which are the basis of almost every flocking algorithm.

We have also categorised the models introduced by various works, as the model constrains the control algorithm, influences the capabilities of the system and is one of the most important factors when designing a control algorithm. The models we have reviewed are based on the same assumption, that the vehicle or flock member is particle-like and it can propel itself in the direction and with the velocity desired based on the used control algorithm. All these models resemble the real world, dealing with increasing complexity in the model, external disturbances and nonlinear intrinsic dynamics. A common shortcoming of the examined solutions is that the vehicles in the flock or the formation may have additional constraints missing from the used models.

Furthermore the various communication methods were examined, and it can be concluded that the vast majority of the solutions assume an almost perfect communication environment, where there are no packet losses, the links are symmetric or the radio range is infinite. This is one of the main reasons why nearly all of these solutions work well only in simulations.

It can also be concluded that only a small fraction of the works from the literature introduce simulations and experiments to study the stability and scalability under realistic environmental conditions. The majority of the solutions are not tested in real use cases, remaining only a mathematical basis for further research. The real use cases were also introduced, providing an overview of the existing simulators and robots which are used to prove the applicability of the theoretical algorithms.

REFERENCES

[1] I. Navarro, *Exploring the split and join capabilities of a robotic collective movement framework*, Int J Adv Robotic Sy, 10 (2013).

[2] A. Abdessameud and A. Tayebi, *On consensus algorithms for double-integrator dynamics without velocity measurements and with input constraints*, Systems & Control Letters, 59 (2010), pp. 812–821.

[3] ———, *On consensus algorithms design for double integrator dynamics*, Automatica, 49 (2013), pp. 253–260.

[4] G. Antonelli, F. Arrichiello, and S. Chiaverini, *Flocking for multi-robot systems via the null-space-based behavioral control*, Swarm Intelligence, 4 (2010), pp. 37–56.

[5] T. C. Aysal, M. E. Yildiz, A. D. Sarwate, and A. Scaglione, *Broadcast gossip algorithms for consensus*, Signal Processing, IEEE Transactions on, 57 (2009), pp. 2748–2761.

[6] M. Bonani, V. Longchamp, S. Magnenat, P. Rtornaz, D. Burnier, G. Roulet, F. Vaussard, H. Bleuler, and F. Mondada, *The MarXbot, a Miniature Mobile Robot Opening new Perspectives for the Collective-robotic Research*, in International Conference on Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ, IEEE International Conference on Intelligent Robots and Systems, IEEE Press, 2010, pp. 4187–4193.

[7] M. Bonani, S. Magnenat, P. Rétornaz, and F. Mondada, *The hand-bot, a robot design for simultaneous climbing and manipulation*, in Intelligent Robotics and Applications, Springer, 2009, pp. 11–22.

[8] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, *Randomized gossip algorithms*, Information Theory, IEEE Transactions on, 52 (2006), pp. 2508–2530.

[9] M. Camperi, A. Cavagna, I. Giardina, G. Parisi, and E. Silvestri, *Spatially balanced topological interaction grants optimal cohesion in flocking models*, Interface focus, 2 (2012), pp. 715–725.

[10] Y. Cao and W. Ren, *Distributed coordinated tracking with reduced interaction via a variable structure approach*, Automatic Control, IEEE Transactions on, 51 (2012), pp. 33–48.

[11] N. Carlési and P. Bianchi, *Distributed coordination of a formation of heterogeneous agents with individual regrets and asynchronous communications*, in Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, IEEE, 2012, pp. 3504–3511.

[12] A. Cavagna, L. Del Castello, I. Giardina, T. Grigera, A. Jelic, S. Melillo, T. Mora, L. Parisi, E. Silvestri, M. Viale, et al., *Flocking and turning: a new model for self-organized collective motion*, arXiv preprint arXiv:1403.1202, (2014).

[13] H. Çelikkanat and E. Şahin, *Steering self-organized robot flocks through externally guided individuals*, Neural Computing and Applications, 19 (2010), pp. 849–865.

[14] L. Cheng, Z.-G. Hou, M. Tan, Y. Lin, and W. Zhang, *Neural-network-based adaptive leader-following control for multiagent systems with uncertainties*, Neural Networks, IEEE Transactions on, 21 (2010), pp. 1351–1358.

[15] J. S. Coleman, *Foundations of social theory*, Harvard University Press, 1994.

[16] M.-C. Fan, Z. Chen, and H.-T. Zhang, *Velocity-free compact rigid flock control with input saturations*, in Control and Decision Conference (CCDC), 2013 25th Chinese, IEEE, 2013, pp. 316–321.

[17] E. Ferrante, W. Sun, A. Turgut, M. Dorigo, M. Birattari, and T. Wenseleers, *Self-organized flocking with conflicting goal directions*, in Proceedings of the European Conference on Complex Systems 2012, Springer, 2013, pp. 607–613.

[18] Z.-G. Hou, L. Cheng, and M. Tan, *Decentralized robust adaptive control for the multiagent system consensus problem using neural networks*, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 39 (2009), pp. 636–647.

[19] S. Housheng and C. Guanrong, *Adaptive flocking with a virtual leader of multiple agents governed by nonlinear dynamics*, Control Conference (CCC), 2010 29th Chinese, (2010), pp. 5827–5832.

[20] A. Jadbabaie, J. Lin, and A. S. Morse, *Coordination of groups of mobile autonomous agents using nearest neighbor rules*, Automatic Control, IEEE Transactions on, 48 (2003), pp. 988–1001.

[21] Z. Jin, W. Xiaoqun, Y. Wenwu, S. Michael, and L. Jun-an, *Flocking of multi-agent dynamical systems based on pseudo-leader mechanism*, Systems & Control Letters, 61 (2012), pp. 195–202.

[22] P. Ke, S. Hou-Sheng, and Y. Yu-Pu, *Coordinated control of multi-agent systems with a varying-velocity leader and input saturation*, Communications in Theoretical Physics, 52 (2009), p. 449.

[23] S.-J. Kim and I.-H. Whang, *Acceleration constraints for maneuvering formation flight trajectories*, Aerospace and Electronic Systems, IEEE Transactions on, 48 (2012), pp. 1052–1060.

[24] A. Kumar, S. Sharma, R. Tiwari, and S. Majumdar, *Area exploration by flocking of multi robot*, Procedia Engineering, 41 (2012), pp. 377–382.

[25] N. E. Leonard and E. Fiorelli, *Virtual leaders, artificial potentials and coordinated control of groups*, in Decision and Control, 2001. Proceedings of the 40th IEEE Conference on, vol. 3, IEEE, 2001, pp. 2968–2973.

[26] X. LI and J. CAO, *Adaptive synchronization for delayed neural networks with stochastic perturbation*, Journal of the Franklin Institute, 345 (2008), pp. 779–791.

[27] H. Lim and C. Kim, *Flooding in wireless ad hoc networks*, Computer Communications, 24 (2001), pp. 353–363.

[28] B. Liu and H. Yu, *Flocking in multi-agent systems with a bounded control input*, in Chaos-Fractals Theories and Applications, 2009. IWCFTA '09. International Workshop on, Nov 2009, pp. 130–134.

[29] I. S. MAGNETIZATION, *The spontaneous magnetization of a two-dimensional ising model*, Physical Review, 85 (1952).

[30] O. Michel, *Webotstm: Professional mobile robot simulation*, arXiv preprint cs/0412052, (2004).

[31] A. Mogilner and L. Edelstein-Keshet, *A non-local model for a swarm*, Journal of Mathematical Biology, 38 (1999), pp. 534–570.

[32] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli, *The e-puck, a robot designed for education in engineering*, in Proceedings of the 9th conference on autonomous robot systems and competitions, vol. 1, IPCB: Instituto Politécnico de Castelo Branco, 2009, pp. 59–65.

[33] D. J. Naffin and G. S. Sukhatme, *Negotiated formations*, in Proceedings of the eighth conference on intelligent autonomous systems, 2004, pp. 181–190.

[34] I. Navarro and F. Matía, *A framework for the collective movement of mobile robots based on distributed decisions*, Robotics and Autonomous Systems, 59 (2011), pp. 685–697.

[35] A. Okubo, *Dynamical aspects of animal grouping: swarms, schools, flocks, and herds*, Advances in biophysics, 22 (1986), pp. 1–94.

[36] R. Olfati-Saber, *Flocking for multi-agent dynamic systems: Algorithms and theory*, Automatic Control, IEEE Transactions on, 51 (2006), pp. 401–420.

[37] J. K. Parrish, S. V. Viscido, and D. Grünbaum, *Self-organized fish schools: an examination of emergent properties*, The biological bulletin, 202 (2002), pp. 296–305.

[38] C. Pinciroli, V. Trianni, R. OGrady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, et al., *Argos: a modular, parallel, multi-engine simulator for multi-robot systems*, Swarm intelligence, 6 (2012), pp. 271–295.

[39] S. A. Quintero, G. E. Collins, and J. P. Hespanha, *Flocking with fixed-wing uavs for distributed sensing: A stochastic optimal control approach*, in American Control Conference (ACC), 2013, IEEE, 2013, pp. 2025–2031.

[40] C. W. Reynolds, *Flocks, herds and schools: A distributed behavioral model*, in ACM SIGGRAPH Computer Graphics, vol. 21, ACM, 1987, pp. 25–34.

[41] J. Roberts, T. Stirling, J.-C. Zufferey, and D. Floreano, *Quadrotor using minimal sensing for autonomous indoor flight*, in Proceedings of the European Micro Air Vehicle Conference and Flight Competition (EMAV2007), 2007.

[42] H. Shi, L. Wang, and T. Chu, *Coordinated control of multiple interactive dynamical agents with asymmetric coupling pattern and switching topology*, in Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, IEEE, 2006, pp. 3209–3214.

[43] ———, *Virtual leader approach to coordinated control of multiple mobile agents with asymmetric interactions*, Physica D: Nonlinear Phenomena, 213 (2006), pp. 51–65.

[44] H. Shi, L. Wang, and T. Chu, *Virtual leader approach to coordinated control of multiple mobile agents with asymmetric interactions*, Physica D, 213 (2006), pp. 51–65.

[45] W. M. Spears, D. F. Spears, J. C. Hamann, and R. Heil, *Distributed, physics-based control of swarms of vehicles*, Autonomous Robots, 17 (2004), pp. 137–162.

[46] H. Su, G. Chen, X. Wang, and Z. Lin, *Adaptive second-order consensus of networked mobile agents with nonlinear dynamics*, Automatica, 47 (20111), pp. 368–375.

[47] H. Su, X. Wang, and Z. Lin, *Flocking of multi-agents with a virtual leader, part i: with a minority of informed agents*, Proc. the 46th IEEE Conference on Decision and Control, (December 2007), pp. 2937–2942.

[48] ———, *Flocking of multi-agents with a virtual leader, part ii: with a virtual leader of varying velocity*, Proc. the 46th IEEE

Conference on Decision and Control, (December 2007), pp. 1429–1434.

[49] H. Su, N. Zhang, M. Z. Chen, H. Wang, and X. Wang, *Adaptive flocking with a virtual leader of multiple agents governed by locally lipschitz nonlinearity*, Nonlinear Analysis: Real World Applications, 14 (2013), pp. 798 – 806.

[50] A. E. Turgut, H. Çelikkanat, F. Gökçe, and E. Şahin, *Self-organized flocking in mobile robot swarms*, Swarm Intelligence, 2 (2008), pp. 97–120.

[51] G. Vásárhelyi, C. Virágh, G. Somorjai, N. Tarcai, T. Szörényi, T. Nepusz, and T. Vicsek, *Outdoor flocking and formation flight with autonomous aerial robots*, arXiv preprint arXiv:1402.3588, (2014).

[52] C. Virágh, G. Vásárhelyi, N. Tarcai, T. Szörényi, G. Somorjai, T. Nepusz, and T. Vicsek, *Flocking algorithm for autonomous flying robots*, Bioinspiration & biomimetics, 9 (2014), p. 025012.

[53] M. Wang, H. Su, M. Zhao, M. Z. Chen, and H. Wang, *Flocking of multiple autonomous agents with preserved network connectivity and heterogeneous nonlinear dynamics*, Neurocomputing, (2013), pp. 169–177.

[54] Q. Wang, H. Fang, J. Chen, Y. Mao, and L. Dou, *Flocking with obstacle avoidance and connectivity maintenance in multi-agent systems*, in Decision and Control (CDC), 2012 IEEE 51st Annual Conference on, IEEE, 2012, pp. 4009–4014.

[55] X. F. WANG, *Complex networks: Topology, dynamics and synchronization*, International Journal of Bifurcation and Chaos, 12 (2002), pp. 885–916.

[56] G. Wen, Z. Duan, H. Su, G. Chen, and W. Yu, *A connectivity-preserving flocking algorithm for multi-agent dynamical systems with bounded potential function*, IET Control Theory & Applications, 6 (2012), pp. 813–821.

[57] W. Yu, J. Cao, and J. Lü, *Global synchronization of linearly hybrid coupled networks with time-varying delay*, SIAM Journal on Applied Dynamical Systems, 7 (2008), pp. 108–133.

[58] W. Yu, G. Chen, and M. Cao, *Distributed leaderfollower flocking control for multi-agent dynamical systems with time-varying velocities*, Systems & Control Letters, 59 (2010), pp. 543–552.

[59] W. Yu, G. Chen, M. Cao, and J. Kurths, *Second-order consensus for multiagent systems with directed topologies and nonlinear dynamics*, IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society, 40 (2010), pp. 881–91.

[60] W. Yu, G. Chen, and J. Lü, *On pinning synchronization of complex dynamical networks*, Automatica, 45 (2009), pp. 429–435.

[61] W. Yu, W. X. Zheng, G. Chen, W. Ren, and J. Cao, *Second-order consensus in multi-agent dynamical systems with sampled position data*, Automatica, 47 (2011), pp. 1496–1503.

[62] H. Zhang, J. Llorca, C. C. Davis, and S. D. Milner, *Nature-inspired self-organization, control, and optimization in heterogeneous wireless networks*, Mobile Computing, IEEE Transactions on, 11 (2012), pp. 1207–1222.

[63] H.-T. Zhang, C. Zhai, and Z. Chen, *A general alignment repulsion algorithm for flocking of multi-agent systems*, Automatic Control, IEEE Transactions on, 56 (2011), pp. 430–435.

[64] Y. Zheng and L. Wang, *Consensus of heterogeneous multi-agent systems without velocity measurements*, International Journal of Control, 85 (2012), pp. 906–914.

[65] J. Zhou, X. Wu, W. Yu, M. Small, and J.-a. Lu, *Flocking of multi-agent dynamical systems based on pseudo-leader mechanism*, Systems & Control Letters, 61 (2012), pp. 195–202.

[66] A.-M. Zou, K. D. Kumar, and Z.-G. Hou, *Attitude coordination control for a group of spacecraft without velocity measurements*, Control Systems Technology, IEEE Transactions on, 20 (2012), pp. 1160–1174.

Edited by: Giacomo Cabri and Emma Hart
Received: Dec 15, 2014
Accepted: Jul 10, 2015

DOI 10.12694/scpe.v16i3.1100
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 16, Number 3, pp. 249–269. http://www.scpe.org © 2015 SCPE

OVERLAY SERVICE COMPUTING - MODULAR AND RECONFIGURABLE COLLECTIVE ADAPTIVE SYSTEMS

EVANGELOS POURNARAS *

Abstract. Distributed software systems that determine virtual communication structures on top of physical networks, the overlay networks, are a well-established approach to build various applications of collective adaptive systems such as peer-to-peer file sharing, multimedia multi-casting, aggregation in distributed databases or routing in wireless sensor networks. Despite the significance of this approach to apply collective adaptive systems in practice, applications based on overlay networks often result in a complex integration of the operational logic with topological management. This approach results in low abstraction, modularity and reconfigurability of applications that require one or more overlay networks to operate. This paper challenges this design approach by introducing the notion of overlay services that provide generic application capabilities of a broad application scope enabled by one or more overlay networks. This paper contributes the multi-level conceptual architecture of ASMA that structures and guides the realisation of overlay services by using only a few lines of high-level algorithmic expressions. Two overlay services realised according to ASMA provide a proof-of-concept for the high abstraction, modularity and reconfigurability achieved in collective adaptive systems based on overlay networks.

Key words: distributed system, overlay network, overlay service, architecture, middleware, abstraction, modularity, reconfigurability

AMS subject classifications. 68M, 68U

1. Introduction.

A plethora of collective adaptive systems and applications are build by software that determines virtual communication structures on top of physical networks such as the Internet, mobile or wireless sensor networks. These structures are usually referred to as ‘overlay networks’ and they represent interaction patterns or linking of information that mandate the operation and optimisation of a distributed application [21, 34]. For example, IP-multicasting is not widely adopted due to economic and technical factors related with security and a high protocol complexity for network service providers [5]. On the contrary, network communication can be structured in an overlay network organised in a tree topology used for multicasting multi-media content in the application-level.

Applications based on overlay networks often result in a complex integration of the operational logic with topological management. Such applications deal with topological and organisational complexity as a way to be dynamic, adaptive and capture information changes during their runtime. For example, the organisation of a tree overlay network with certain topological properties shall improve the performance of multimedia multicasting, e.g., tree balancing and constraints in node degrees [34]. This integrating design approach results in low abstraction, modularity and reconfigurability of applications that require one or more overlay networks to operate.

The design complexity of overlay networks has also raised an argument about the impact of overlay networks on the future development of the Internet and its distributed applications. There are two main opposing views in this argument: The *purists* view overlay networks as testbeds used for the implementation and experimentation of novel Internet architectures. Purists do not view overlay networks as viable or coexisting architectural elements of the future Internet. In contrast, *pluralists* envision overlay networks as a possible solution to deal with the heterogeneity of applications and the business challenges of network service providers related to the development and adoption of technological innovations in the Internet infrastructure. This paper reasons about “*a philosophical revolution in how developers use overlays, rather than a technical alteration in how they build them*” [1]. Given that applications of collective adaptive systems based on overlay networks emerge faster than the adoption of overlay networks in the Internet infrastructure, it is evident that a higher abstraction, modularity and reconfigurability for services of overlay network is required as identified in earlier work [11, 17, 20, 18, 33, 4, 16, 41, 42].

This paper introduces the notion of an ‘overlay service’ that is a distributed stand-alone software system, e.g., middleware, that is based on one or more overlay networks and provides generic application capabilities

*Professorship of Computational Social Science, ETH Zurich, Zurich, Switzerland (epournaras@ethz.ch).

of a broad application scope. This paper contributes ASMA, the *Adaptive Self-organisation in a Multi-level Architecture* that is a new conceptual multi-level architecture to design and prototype overlay services [25]. A higher abstraction, modularity and reconfigurability are these novel qualitative properties that collective adaptive system inherit when designed according to ASMA. The applicability of the ASMA architecture is challenged by illustrating the modularity and reconfigurability in two architectural realisations of overlay services. These realisations concern earlier work on large-scale networked systems that perform highly complex functionality in a fully decentralised and collective fashion. However, in the new context of this paper it is shown how only a few lines of high-level algorithmic logic defined by ASMA guide and unravel the design and prototyping process of these complex adaptive systems. In addition, experimental evaluation of the overlay services provides a proof-of-concept for the high abstraction, modularity and reconfigurability achieved with the ASMA architecture. Yet, evaluation also dissects the performance trade-offs made as a result of introducing generic collective adaptive systems.

This paper is outlined as follow: Section 2 defines the main concepts proposed in this paper. Section 3 illustrates an overview of the ASMA architecture. Section 4 introduces the three levels of the ASMA architecture and their interactions. Section 5 shows how overlay services can be realised by the ASMA architecture. In the same section, the high abstraction, modularity and reconfigurability of the ASMA overlay services are studied experimentally. Section 6 compares ASMA with related work. Finally, Section 7 concludes this paper and outlines future work.

2. Overlay Networks and Services. In the context of this paper, a collective adaptive system of overlay networks consists of the following computing entities[1] as illustrated in Figure 2.1:

1. *Host*: This entity is a physical machine with a network interface connected to a physical network.
2. *Peer*: This entity is a software environment that hosts agents and enables their communication.
3. *Agent*: This entity is a software system that carries out, with some degree of independence or autonomy, a set of operations defined by a distributed application.
4. *Node*: This entity is a logical abstraction and representation of an agent in an overlay network.

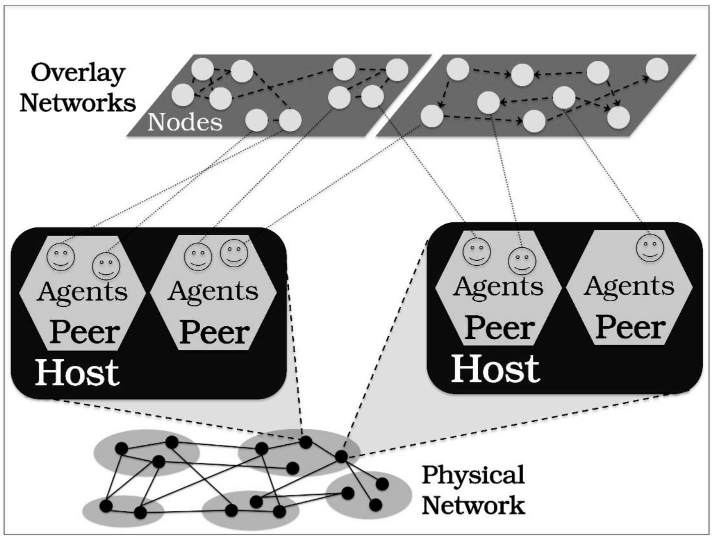


FIG. 2.1. *The four entities defined within the context of collective adaptive systems built with overlay networks: (i) host, (ii) peer, (iii) agent and (iv) node.*

An *overlay network* is defined in this paper as a graph representation of information managed by the agents of a collective adaptive system. Peers and agents can be part of a middleware system or integral parts of

[1]These overloaded terms may be used in different ways in various computing areas such as middleware, peer-to-peer, multi-agent and telecommunication systems. For example, overlay networks are built by virtual nodes that appear to be related to 'peers', 'agents', 'hosting machines' or 'software clients' in literature.

distributed applications. As shown in Figure 2.1, a host may contain more than one peer that each may also contain multiple agents. Finally, note that an overlay network is defined by agent memberships: Every agent in a network stores unique network identifiers and other information of other agents in a limited (partial) set. The memberships known to an agent are its partial *view* of the system.

This paper studies collective adaptive systems of overlay networks designed to serve a wide range of distributed applications. These generic systems are referred to in this paper as overlay services[2]. An *overlay service* is defined as a decentralised software system that provides a number of collective intelligence capabilities of a broad application scope enabled by one or more overlay networks. Overlay services can be realised as distributed middleware systems. Section 5 illustrates two representative examples of overlay services: (i) self-organisation of overlay networks in tree topologies [30] and (ii) aggregation of dynamically changing information distributed in the network [29].

An overlay service is characterised by its quality. The *quality of an overlay service* is defined as a measurable metric that quantifies the degree to which this overlay service can meet certain application objectives [19], for instance, the average response time [22] of queries to a directory service that relies on an overlay network.

3. An Architecture for Overlay Services. This paper introduces ASMA, the Adaptive Self-organisation in a Multi-level Architecture. ASMA is a conceptual self-organisation architecture with which different generic overlay services can be designed. The implementation of an individual overlay service in the ASMA architecture is referred to as *architectural realisation*. An architectural realisation entails the realisations of tasks defined in ASMA.

ASMA addresses the challenges of abstraction, modularity and reconfigurability in overlay services by introducing (i) a multi-level architecture and (ii) inter-level interactions. The complexity of a collective adaptive system is managed by multiple application-independent levels, each with a specific self-organisation goal. Each level in ASMA supports the level above and configures the level below. These bottom-up and top-down inter-level interactions tune the self-organisation operations in each level to improve the quality of an overlay service.

Figure 3.1 illustrates the ASMA architecture positioned in a single peer. The design of an overlay service in ASMA is defined by sets of (i) *criteria* and (ii) *samples* within three application-independent reconfigurable self-organisation levels: (i) the *discovery level*, (ii) the *structuring level* and (iii) the *coordination level*. A *criterion* is runtime feedback information that parametrises the operation of a level. A *sample* is continuously updated information required for the operation of a level in the ASMA architecture. For each set of *criteria* at each level in the architecture, a set of *samples* is generated. The *discovery level* discovers required information in the network. The *structuring level* structures this information. Finally the *coordination level* uses the structured information to build and provide the intended functionality to an application.

Each level of this architecture is managed by one or more autonomous agents. Two corresponding levels in two different peers are able to communicate remotely. In other words, agents of the same type are able to remotely interact. ASMA defines two types of interactions: (i) vertical and (ii) horizontal. A *vertical interaction* is the (local) exchange of *criteria* and *samples* between two different levels of ASMA located within the same peer. In contrast, a *horizontal interaction* is the (remote) exchange of *criteria* and *samples* between the same two levels of ASMA located within two different peers.

Criteria are provided in a top-down fashion in vertical interactions: from each level to the level below. In horizontal interactions, *criteria* are provided by a remote corresponding level. A *criterion* is generated based on some given *samples*. For example, *criteria* can be generated by a change in the value of a monitored metric, changes in the underlying network or the result of an agent negotiation. ASMA defines three types of *criteria* in its vertical interactions:

- *Organisational criteria*: These are *criteria* that parametrise the self-organisation operation of an overlay service. They are externally provided by an application as input to the *coordination level*.

[2]In contrast to overlay services as defined in this section, *service overlay networks* (SONs), introduced by [39], refer to a number of dedicated hosts in ISPs that explicitly allocate resources for peer-to-peer or other decentralised systems. Bandwidth resources of certain quality are provisioned based on SLAs. SONs provide a generic model for the allocation of Internet resources to decentralised systems and applications rather than a methodology of how to provide generic application capabilities enabled by overlay networks. Note that this distinction is identified in earlier work [12] that refers to overlay services as *overlay-based services*. Overlay services can coexist on top of SONs. In this case, SONs provide a business model for the Internet resource allocation required for overlay services and their applications [6].

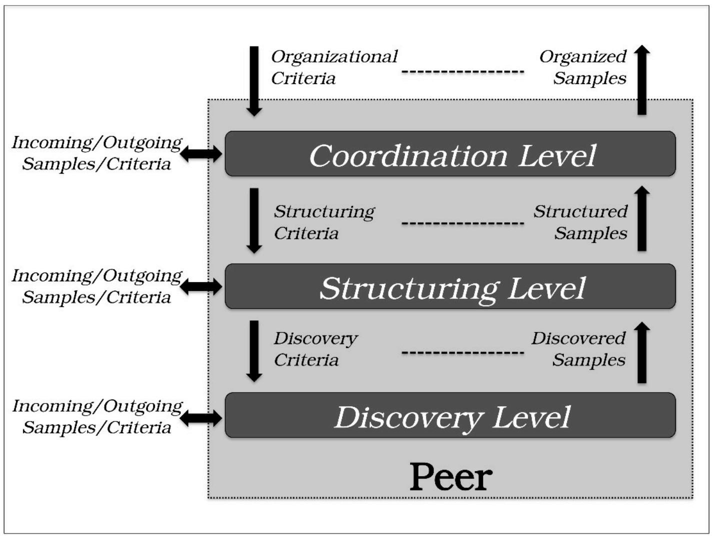


FIG. 3.1. *The three levels of the ASMA architecture in a peer.*

- *Structuring criteria*: These are *criteria* that parametrise the structuring of *samples* to improve the quality of an overlay service. They are provided by the *coordination level* to the *structuring level*.
- *Discovery criteria*: These are *criteria* that parametrise the dissemination and collection of *samples* in a network. They are provided by the *structuring level* to the *discovery level*.

Samples are provided in a bottom-up fashion in vertical interactions: from each level to the level above. In horizontal interactions, *samples* are provided by a remote corresponding level. A *sample* is defined within the context of an overlay service and may represent a wide range of information related to any of the entities of Figure 2.1: from information about the local host, such as its IP address or its geographic location, to information about the user, such as his/her reputation and trust in an online community. This information is usually abstracted from the application. For example, the reputation of a user in an online community can be represented as an abstract rank value of a node in the overlay network of this community. Three types of *samples* are defined in the vertical interactions of ASMA:

- *Discovered samples*: These are *samples* discovered in the network that are locally provided to the *structuring level* in which they are managed.
- *Structured samples*: These are *discovered samples* required for building an overlay service. They are provided by the *structuring level* to the *coordination level*.
- *Organised samples*: These are the output *samples* achieving a certain quality of an overlay service. They are provided by the *coordination level* to applications.

The *criteria* and *samples* illustrated above are application-independent and are abstracted from the application. *Samples* are provided from the one level to the other if a condition is satisfied. *Criteria* are feedback parametrisation triggered by the consumption of *samples* in a level of the architecture. The *criteria* and *samples* exchanged in horizontal interactions are referred to as (i) *incoming criteria*, (ii) *incoming samples*, (iii) *outgoing criteria* and (iv) *outgoing samples*. The semantic of these *samples* and *criteria* is defined by an architectural realisation.

In the rest of this paper, the ASMA architecture and its realisations are illustrated via high level algorithmic expressions that aim at guiding the prototyping process and structuring the different interactive levels in an overlay service and their interactions[3].

[3]The exact algorithms concerning the functionality of the overlay services are out of the scope of this paper and are illustrated in earlier work about these overlay services.

4. Architectural Levels. Each individual level of ASMA is defined according to Figure 4.1. Algorithms 1 and 2 provide an abstraction for the event generations and reactions in each ASMA level. A realisation of these algorithms is illustrated in Algorithms 3 and 4 of Section 4.1. Assume an arbitrary ASMA level that (i) generates some arbitrary output *criteria* and *samples* and (ii) reacts to some arbitrary input *criteria* and *samples*. Input *criteria* trigger execution of the `adapt` task that generates the output *samples*. Similarly, input *samples* trigger execution of the `consume` task that generates output *criteria*. The `provide` task sends the output *samples* to the level above and the `configure` task sends the output *criteria* to the level below. *Samples* and *criteria* can also be sent to a remote corresponding level in a horizontal interaction. However, the `provide` task may call the `consume` task of the same level instead of the one in the level above, suggesting in this way the possibility of internal feedback loops within each level. Similarly, the `configure` task may call the `adapt` task of the same level instead of the one in the level below. These internal calls, within a level, are possible options defined within a realisation of an ASMA level. The realisation of a level is defined by the implementation and scheduling of its tasks.

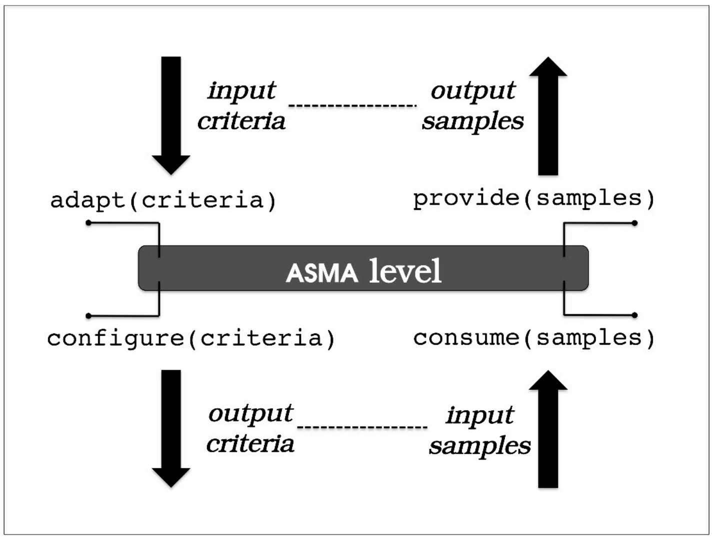


FIG. 4.1. *The executed tasks of an abstract ASMA level.*

Algorithm 1 Generations of output events in an ASMA level.

```
1: while a condition is satisfied do
2:     provide(samples)
3: end while
Ensure: output
```

Algorithm 2 Reactions of input events in an ASMA level.

```
Require: input
1: if input=criteria then
2:     samples=adapt(criteria)
3: else // input=samples
4:     criteria=consume(samples)
5:     configure(criteria)
6: end if
```

The three levels of ASMA are summarised as follows: The *discovery level*, positioned at the bottom of the architecture, performs discovery of remote *samples* required by an overlay service. *Sample* discovery is achieved by horizontal interactions that disseminate *outgoing criteria* and trigger the remote collection of *incoming samples* from the network. The dissemination and collection of *samples* is parametrised by the *discovery* *criteria* received from the *structuring level*. The *structuring level* structures the *discovered samples* provided by the *discovery level* or other *incoming samples* received by horizontal interactions. The *structuring criteria* customise the structuring and selection of the *structured samples* provided to the *coordination level*. Finally, the *coordination level* coordinates the main functionality of an overlay service according to a set of *organisational criteria*. The *coordination level* uses the *structured samples* or other *incoming samples* received by horizontal interactions to update the *organised samples* provided to an application. The quality of an overlay service achieved with certain *structured samples* is evaluated resulting in a new set of *structuring criteria* that improve *structured samples*. The *organised samples* are a result of a continuous inter-level interactions and adaptations between the three levels of ASMA.

To certain extent, the multi-level architecture of ASMA resembles the simple architectural principle of OSI layering. However, collective adaptive systems built at the application level with overlay networks continue to integrate topological management with their main operational logic resulting in limited abstraction, modularity and reconfigurability. Therefore an OSI-like architectural principle with self-adaptive levels [43, 44] is a promising novel design approach to tackle this challenge. The modularity of the ASMA architecture could be extended to more than three levels assembled in various architectural patterns as shown in earlier work [32]. Each architectural level of ASMA is abstracted as shown in Figure 4.1, therefore, more levels could be added in the ASMA stack. This paper focuses on three levels as the overlay service realisations illustrated in Section 5 show empirically that three levels are a cost-effective compromise to model the complexity of collective adaptive systems operating in decentralised networked environments. The rest of this section illustrates each level of the ASMA architecture in detail.

4.1. The *discovery level*. The *discovery level* is responsible for the distribution and availability of *samples* to every peer of the hosts of a network providing the abstraction of *sample* discovery in the *structuring level*. The dissemination of *outgoing samples* in the network is performed using *outgoing criteria* and the `configure` task. Symmetrically, the collection of remote *incoming samples* triggers the `consume` task. Furthermore, the execution of the `adapt` task is triggered by *incoming criteria* and by the *discovery criteria* that configure the dissemination and collection of *samples* in favour of the *structuring level*. Remote communication between peers is possible as the *samples* disseminated and collected in the network contain routing information, e.g. the IP address and port number.

Making distributed *samples* locally available to the peers of a network is challenging and crucial for building decentralised overlay services. Middleware systems based on centralised information lookup or distributed lookup mechanisms designed with specific applications types in mind cannot always support scalable and generic overlay services. ASMA introduces the *discovery level* in the foundations of the architecture to bridge the information gap of decentralisation in overlay networks.

This paper focuses on a gossip-based realisation of the *discovery level*[4]. Gossiping is a simple and generic probabilistic communication model according to which agents exchange *samples* in a 'push', 'pull', or 'push-pull' fashion [14]. The exchange of *samples* is random to certain degree but other more intelligent policies can be applied as well. Gossiping information is spread in an epidemic fashion within a network [36, 35]. Furthermore, gossiping is able to prevent clustering of a network by cascading failures of its hosts.

Algorithm 3 and 4 illustrate a high-level description of a 'push-pull' gossiping protocol that realises the horizontal interactions of the *discovery level*. The *outgoing criteria* represent a 'push' message and the *incoming samples* represent a 'pull' message. Both contain local *discovered samples* exchanged in a gossiping fashion. A gossip-based *discovery level* periodically sends *outgoing criteria* to a remote *discovery level* of a selected peer defined within these *criteria* (line 3 of Algorithm 3). The *discovered samples* are also provided periodically to the *structuring level* (line 2 of Algorithm 3). Furthermore, the *discovery level* reacts to *incoming criteria* by adapting its *discovered samples* and providing in return *outgoing samples* (line 2-3 of Algorithm 4). The *incoming samples* are consumed and generate the next *outgoing criteria* (line 5 of Algorithm 4).

The core gossiping operations are performed in the `adapt` and `consume` tasks illustrated in Algorithm 5 and 6. The `adapt` task (i) handles *incoming criteria* that are actual 'push' gossiping messages and (ii) generates

[4]Gossiping is chosen because of its robustness properties and the rapid dissemination of information in distributed networks. These properties benefit the overlay services studied in this paper. Other mechanisms can be employed as well, e.g., flooding [15], random walks [8] and DHT overlays [38].

Algorithm 3 Event generations by a gossip-based *discovery level*.

```
loop // periodically
    provide(discovered samples)
    configure(outgoing criteria)
end loop
Ensure: output
```

Algorithm 4 Event reactions by a gossip-based *discovery level*.

```
Require: input
if input=incoming criteria then
    outgoing samples=adapt(incoming criteria)
    provide(outgoing samples)
else // input=incoming samples
    outgoing criteria=consume(incoming samples)
end if
```

outgoing samples that are actual ‘pull’ gossiping messages (line 1-5 of Algorithm 5). The *discovery criteria* parametrise gossiping by, for example, selecting policies [14] that tune the dissemination of *samples* under various network conditions, e.g. failures in hosts. The `consume` task updates the *discovered samples* with *incoming samples* (line 1 of Algorithm 6) and selects *outgoing samples* to disseminate from the *discovered samples* (line 2 of Algorithm 6).

Algorithm 5 The `adapt` task in a gossip-based *discovery level*.

```
Require: criteria
if criteria=incoming criteria then
    outgoing samples=selectToDisseminate(discovered samples)
    incoming samples=getSamples(incoming criteria)
    outgoing criteria=consume(incoming samples)
    return outgoing samples
else // criteria=discovery criteria
    // Parametrises gossiping [14]:
    // ‘peer selection’, ‘view propagation’ and ‘view selection’ policies
    return discovered samples
end if
Ensure: samples
```

Algorithm 6 The `consume` task in a gossip-based *discovery level*.

```
Require: incoming samples
discovered samples=selectToCollect(incoming samples)
outgoing samples=selectToDisseminate(discovered samples)
outgoing criteria=getCriteria(outgoing samples)
return outgoing criteria
Ensure: outgoing criteria
```

Finally, the `selectToCollect` and `selectToDisseminate` tasks (line 1 and 2 of Algorithm 6) implement the push-pull gossiping interactions and correspond to the respective selection tasks of the peer sampling service [14]. The `getCriteria` task performs the selection of the agent to gossip with (line 3 of Algorithm 6) and the `getSamples` task simply derives the *incoming samples* included in a set of *incoming criteria* (line 3 of Algorithm 5).

4.2. The *structuring level*. The *structuring level* is responsible for the management of *discovered samples*, providing in this way an abstraction to the *coordination level*. More specifically, the *structuring level* performs (i) structuring, such as sorting, clustering and classification of the *discovered samples* received from the *discovery level* and (ii) selection of the *structured samples* provided to the *coordination level*.

Structuring and selection are based on criteria defined by an adaptation strategy. The following three examples illustrate some adaptation strategies:

- Sorting a list of ranked *samples* in an ascending order and selecting the first *sample* from the list.
- Clustering a set of ranked *samples* based on their ranking distance and selecting the highest or lowest ranked *sample* in each cluster.

- Classifying a set of *samples* in a number of classes and selecting the most recently added *sample* from each class.

There is a wide range of adaptation strategies that can be designed regarding a certain self-organisation goal of an overlay service. Adaptation strategies provide dynamic management of *samples* as:

- Multiple adaptation strategies can be adopted dynamically during runtime.
- The parameters of an adaptation strategy that define the structuring and selection of *samples* can be reconfigured during runtime.

Both approaches are based on feedback received within the *structuring criteria* from the *coordination level*. The *structuring criteria* result in new *structured samples* that potentially improve the quality of an overlay service. In this case, the *structuring criteria* are an actual feedback about the provided *structured samples* that can be used by the *structuring level* for either (i) switching to a different adaptation strategy or (ii) reconfiguring the parameters that define a certain adaptation strategy.

Algorithms 7 and 8 illustrate the event generations and reactions in the *structuring level*. The tasks executed by the *structuring level* are specialisations of an abstract ASMA level. Optionally, *criteria* and *samples* can be exchanged via horizontal interactions.

Algorithm 7 Event generations by the *structuring level*.

```
1: while a condition is satisfied do
2:     provide(structured samples)
3:     provide(outgoing samples) // Optional, defined in a level realisation
4:     configure(outgoing criteria) // Optional, defined in a level realisation
5: end while
Ensure: output
```

Algorithm 8 Event reactions by the *structuring level*.

```
Require: input
1: if input=structuring criteria then
2:     structured samples=adapt(structuring criteria)
3: else if input=discovered samples then
4:     discovery criteria=consume(discovered samples)
5:     configure(discovery criteria)
6: else // input=incoming criteria or incoming samples
7:     // Optional, defined in a level realisation
8: end if
```

Algorithms 9 and 10 illustrate the `adapt` and `consume` task in the *structuring level*. The `adapt` task is based on two subtasks, the `adopt` and `selectToProvide`. The first subtask is responsible for the choice and reconfiguration of the adaptation strategy based on which the *structured samples* are selected (line 2 of Algorithm 9). A learning or rule-based system may be used to correlate certain feedback information contained in the *structuring criteria* with a number of adaptation strategies supported by the *structuring level*. Furthermore, the `selectToProvide` subtask selects a number of *structured samples* provided to the *coordination level* (line 3 of Algorithm 9). The selection of *structured samples* is performed based on criteria defined within the adopted adaptation strategy. The `consume` task defines the structuring of the *discovered samples*, such as sorting, clustering, classification, etc., based on the adopted adaptation strategy (line 2 of Algorithm 10). The `adopt`, `selectToProvide` and `structure` subtasks are realised by each overlay service and therefore their definition is subject of an architectural realisation

Algorithm 9 The `adapt` task in the *structuring level*.

```
Require: criteria
1: if criteria=structuring criteria then
2:     strategy=adopt(structuring criteria)
3:     structured samples=selectToProvide(strategy)
4:     return structured samples
5: else if criteria=incoming criteria then
6:     // Optional, defined in a level realisation
7: end if
Ensure: structured samples
```

Algorithm 10 The `consume` task in the *structuring level*.

```
Require: samples
1: if samples=discovered samples then
2:     discovery criteria=structure(strategy, discovered samples)
3:     return discovery criteria
4: else if samples=incoming samples then
5:     // Optional, defined in a level realisation
6: end if
Ensure: discovery criteria
```

Adaptation strategies introduce a modularity level in the design phase of an overlay service. By adopting multiple adaptation strategies or by reconfiguring a certain strategy, self-organisation provides a flexible compositional environment for meeting complex organisational goals related to various criteria of complex adaptive systems.

4.3. The *coordination level*. The *coordination level* is responsible for the continuous organisational update of the *organised samples* provided to an application. The update of the *organised samples* is based on the *structured samples* provided by the *structuring level* and the *organisational criteria* provided by an application. Updating the *organised samples* may require some coordination between remote agents of the *coordination level*. These agents are defined within the *structured samples*. Coordination may concern the exchange of *samples* required for the operation of an overlay service, a negotiation between two agents about their required *samples*, a query, or some other type of remote interaction and operation.

The *organised samples* can be tuned by a fitness function [23] or another evaluation scheme that maximises the quality of an overlay service. This process generates a set of *structuring criteria* containing feedback for the *structuring level* to trigger the next *structured samples* that improve the quality of an overlay service. Therefore, the exchange of *samples* and *criteria* between the *structuring level* and the *coordination level* is a continuous and iterative optimisation process of the quality of an overlay service.

Algorithm 11 illustrates the event generations in the *coordination level*. The delivery of the *organised samples* to the application is governed by the *organisational criteria* (line 1 and 2 in Algorithm 11). The *organisational criteria* related with this delivery may concern a certain quality of an overlay service, or an elapsed runtime period.

Algorithm 11 Event generations by the *coordination level*.

```
1: while a condition is satisfied do
2:     provide(organised samples)
3: end while
Ensure: organised samples
```

Algorithm 12 shows the event reactions in the *coordination level*. A simple coordination scenario of horizontal interactions is assumed in which an agent of the *coordination level* sends a set of *outgoing criteria* containing some *outgoing samples* and receives back *incoming samples*. This scenario corresponds to coordination based on an information exchange. Consuming *structured samples* triggers the *outgoing criteria* (line 2 in Algorithm 12) and a set of *incoming criteria* results in providing *outgoing samples* to the agent from which these *criteria* are received (line 8 and 9 in Algorithm 12). Receiving *incoming samples* completes the coordination by sending a set of *structuring criteria* to the *structuring level* (line 5 and 6 in Algorithm 12). Finally, the *organisational criteria* adapt the *organised samples* (line 11 in Algorithm 12) by parametrising the operation of an overlay service.

Algorithm 13 illustrates the `adapt` task. The `adapt` task performs the parametrisation of the *coordination level* as defined in the *organisational criteria* (line 2 in Algorithm 13). It also handles coordination by (i) consuming the *incoming samples* contained in a set of received *incoming criteria*, (ii) configuring the *structuring level* with the *structuring criteria* and (iii) generating the *outgoing samples* that is sent back to the agent that initiates coordination (line 4-9 in Algorithm 13).

The `consume` task, illustrated in Algorithm 14, (i) initiates the coordination by generating *outgoing criteria* (line 2 in Algorithm 14) and (ii) organises the *organised samples* provided to the application (line 5 in

Algorithm 12 Event reactions by the *coordination level*.

```
Require: input
 1: if input=structured samples then
 2:     outgoing criteria=consume(structured samples)
 3:     configure(outgoing criteria)
 4: else if input=incoming samples then
 5:     structuring criteria=consume(incoming samples)
 6:     configure(structuring criteria)
 7: else if input=incoming criteria then
 8:     outgoing samples=adapt(incoming criteria)
 9:     provide(outgoing samples)
10: else // input=organisational criteria
11:     organised samples=adapt(organisational criteria)
12: end if
```

Algorithm 13 The `adapt` task in the *coordination level*.

```
Require: criteria
 1: if criteria=organisational criteria then
 2:     organised samples=parameterise(organisational criteria)
 3:     return organised samples
 4: else if criteria=incoming criteria then
 5:     incoming samples=getSamples(incoming criteria)
 6:     structuring criteria=consume(incoming samples)
 7:     configure(structuring criteria)
 8:     outgoing samples=finaliseCoordination(structuring criteria)
 9:     return outgoing samples
10: end if
Ensure: samples
```

Algorithm 14). This task results in the *structuring criteria* provided to the *structuring level* as feedback for the improvement of the quality of an overlay service.

Algorithm 14 The `consume` task in the *coordination level*.

```
Require: samples
 1: if samples=structured samples then
 2:     outgoing criteria=initialiseCoordination(structured samples)
 3:     return outgoing criteria
 4: else // samples=incoming samples
 5:     structuring criteria=organise(incoming samples)
 6:     return structuring criteria
 7: end if
Ensure: criteria
```

The `adapt` and `consume` tasks show that coordination is an actual response of *outgoing samples* to a set of *incoming criteria* based on the *structuring criteria* (line 8 in Algorithm 13) and *structured samples* (line 2 in Algorithm 14) respectively. Note that the `initialiseCoordination`, `finaliseCoordination`, `organise` and `parameterise` subtasks are realised within an overlay service.

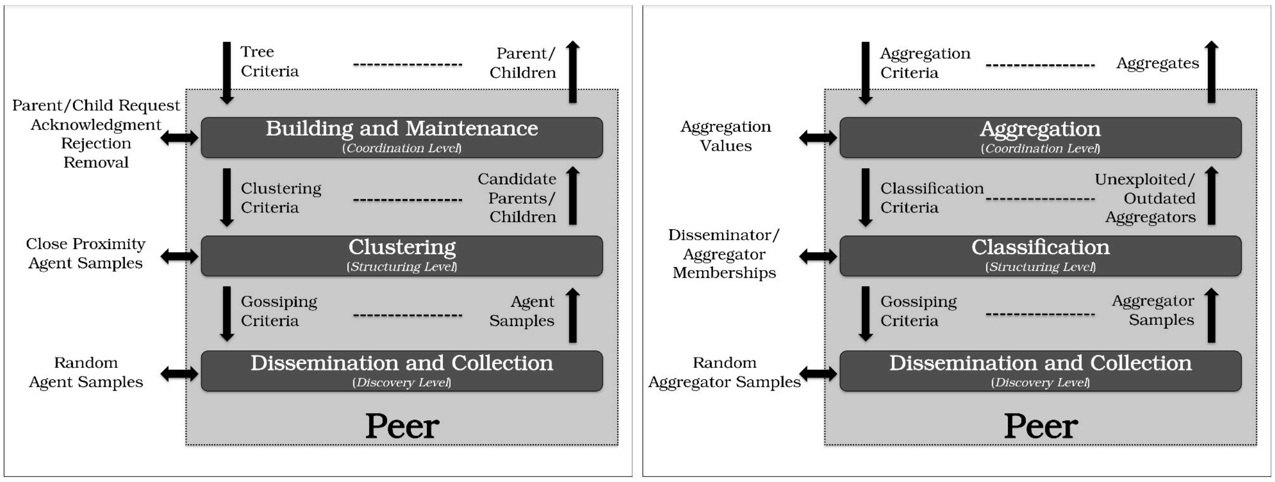
5. Architectural Realisations.

This section illustrates two overlay services designed and realised according to the ASMA architecture:

- AETOS, the Adaptive Epidemic Tree Overlay Service [30].
- DIAS, the Dynamic Intelligent Aggregation Service [29].

These two overlay services have a generic application scope, yet, earlier work illustrates evidence about their applicability in the domain of demand-side energy management [25, 28, 31]. Figure 5.1 illustrates the realized architectures of these overlay services. Both architectures follow the design paradigm of ASMA illustrated in Figure 3.1.

AETOS self-organises overlay networks in various tree topologies to meet different application requirements. Trees are used for operations such as decision-making, aggregation, information dissemination etc., with an applicability in a wide range of distributed applications, e.g., distributed databases [40, 9] and multimedia multicasting [34]. The three levels of ASMA are relevant to model the complexity of such an overlay service: (i) discovery for accessing every possible peer in the network, (ii) structuring and selecting candidate parents/children according to the intended built topology and (iii) coordination between the parents/children



(a) The AETOS overlay service (b) The DIAS overlay service.

FIG. 5.1. *Two overlay service realisations of the ASMA architecture.*

to form bidirectional links. Building and maintenance of a tree topology is entirely performed by the AETOS overlay service without involvement of applications that use it.

DIAS computes aggregates, such as AVERAGE, SUMMATION, MAXIMUM, and STANDARD DEVIATION, of dynamically changing values distributed in every peer of an unstructured overlay network. Collective computation of aggregates benefits a wide range of distributed applications as aggregates are used for load-balancing [26], data mining [37], sensor networks [3] and other. Accurate computations of aggregates are achieved as values that are duplicate or locally changing are detected within the three-level architecture of ASMA: (i) discovery of information from every possible peer, (ii) structuring this information as exploited (duplicate information), unexploited (new information) or outdated (changed information) and (iii) using this information for coordinating accurate computations of aggregates between peers. Periodic computations of system-wide aggregate information are entirely performed by the DIAS overlay service without involvement of applications that use it.

This paper shows how the collective adaptive processes of each of AETOS and DIAS are dissected and realised within the defined tasks of the ASMA architecture. In other words, these overlay services are used as case studies of ASMA. Although the main functionality of AETOS and DIAS is illustrated in earlier work, this paper contributes the novel design approach of ASMA that justifies the higher modularity and reconfigurability in both of these overlay services. This section recalls experimental results of AETOS and DIAS to show how the design methodology of ASMA empirically justifies the high modularity and reconfigurability of these overlay services. Therefore, the earlier experimental results provide new findings in the new context of this paper. These results validate the reconfigurability of overlay services realised according to ASMA architecture and they also provide evidence of how the modularity concept of ASMA is applied in practice. For example, it is shown how results about the performance trade-offs observed in AETOS and DIAS are justified by the reconfigurability engineered in ASMA. Similarly, the performance results of the adaptation strategies provide evidence that overlay services can provide a broad range of modular application capabilities.

The overlay services are implemented and evaluated in the Protopeer prototyping toolkit according to the ASMA architecture [7]. Protopeer is set up to simulate networks of 1500 peers. Each peer hosts three agents[5] that implement the architectural levels of ASMA. Each experiment has a duration calculated in *epochs*. Each epoch in Protopeer lasts 1000 ms.

5.1. AETOS: The Adaptive Epidemic Tree Overlay Service. The *discovery level* of AETOS is realised by the gossiping protocol of the peer sampling service [14] as illustrated in Section 4.1. Gossiping guarantees that a tree topology can reconnect after single node failures as agents continuously exchange information

[5]The *peerlets* of Protopeer implement the agents of each ASMA level.

to discover each other. The nodes of AETOS are ranked with a weight to order their positioning in the formed tree. The weight of the nodes, together with the IP address and the port number, are part of the *discovered samples*.

The *structuring level* clusters *discovered samples* received from the *discovery level* into a *proximity view* $\mathbf{v}_i(proximity)$ that is a list of candidate parents and children. Eight adaptation strategies [27] define criteria for clustering based on the proximity of the node, e.g., minimum or maximum euclidean distance of their weights. In addition, adaptation strategies periodically select and provide the candidate parent and children with the closest proximity to the *coordination level*. Their quality is evaluated by the *coordination level* and feedback is provided to the *structuring level* via *organisational criteria*. Event generations of the *structuring level* are time-based as shown in line 1 of Algorithm 7.

Clustering is tuned by employing the T-MAN mechanism [13], that introduces horizontal interactions between the agents of the *structuring level*. The main intuition behind the introduction of T-MAN is to improve the quality of clustering by letting close proximity agents exchange their *discovered samples* in which they share interest. Algorithms 15 and 16 illustrate the T-MAN functionality embedded in the `adapt` and `consume` task of the *structuring level*. The functionality of T-MAN is injected in the optional blocks of Algorithm 9 and 10. Furthermore, the event generations of T-MAN are periodic, meaning that the `configure`(*outgoing criteria*) at line 4 of Algorithm 7 is executed periodically.

Algorithm 15 T-MAN functionality embedded in the `adapt` task of the *structuring level* in AETOS.

```
Require: criteria
 1: if criteria=incoming criteria then
 2:     incoming samples=getSamples(incoming criteria)
 3:     discovery criteria=structure(strategy, incoming samples)
 4:     configure(discovery criteria)
 5:     outgoing samples=selectToCluster($\mathbf{v}_i(proximity)$)
 6:     return outgoing samples
 7: end if
Ensure: outgoing samples
```

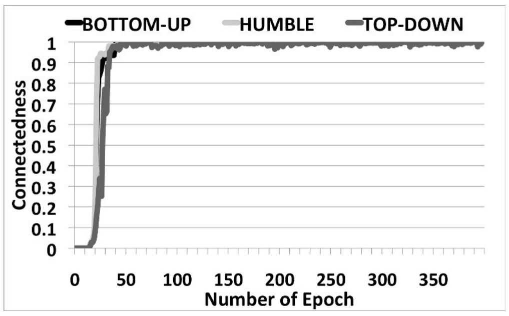
Outgoing criteria and *outgoing samples* contain the exchanged proximity views. The `selectToCluster` subtask selects the ranked node with which the proximity views are exchanged. It also embeds the proximity view in the *outgoing samples* (line 5 and 3 of Algorithm 15 and 16). The `structure` subtask (line 3 and 2 of Algorithm 15 and 16) fills the proximity view according to the adopted adaptation strategy [27].

Algorithm 16 T-MAN functionality embedded in the `consume` task of the *structuring level* in AETOS.

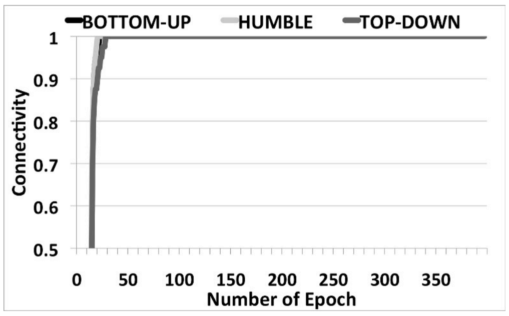
```
Require: samples
 1: if samples=incoming samples then
 2:     discovery criteria=structure(strategy, incoming samples)
 3:     outgoing samples=selectToCluster($\mathbf{v}_i(proximity)$)
 4:     outgoing criteria=getCriteria(outgoing samples)
 5:     return discovery criteria
 6: end if
Ensure: discovery criteria
```

The *coordination level* is responsible for building and maintaining the tree topology. Agents negotiate the formation of bidirectional links with the candidate parents and children received from the *structuring level*. Four types of messages are exchanged that realise horizontal interactions between the agents of the *coordination level*: *request*, *acknowledgment*, *rejection* and *removal*. The *request/removal* messages are used as *outgoing criteria* and the *acknowledgment/rejection* messages as *outgoing samples*. In both cases, the weights of the communicating agents are included using the `getCriteria` and `getSamples` subtasks. The establishment of a link results in *structuring criteria* with a positive feedback to the *structuring level*. In contrast, the rejection of a *request* or the *removal* of a link results in negative feedback. Based on this feedback, the clustering performed in the *structuring level* is tuned to provide candidate parents and children that improve the performance of AETOS [30]. The protocol interactions are realised within the `initialiseCoordination` and `finalise-Coordination` subtasks of Algorithm 14 and 13. Finally, the `organise` subtask of Algorithm 14 facilitates the coordination logic by reacting to the received messages.

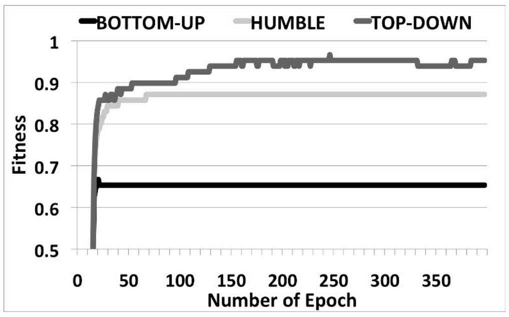
Figure 5.2 illustrates performance trade-offs between three adaptation strategies of AETOS[6]: BOTTOM-UP, HUMBLE and TOP-DOWN. Four performance metrics are shown that measure the quality of the AETOS overlay service: (i) *connectedness*, (ii) *connectivity*, (iii) *fitness* and (iv) *communication cost*. Connectedness measures how well connected nodes are in a single tree, whereas, connectivity measures the extent to which nodes establish the maximum number of links (children) that their resources allow. Fitness evaluates the sorting of the nodes in a tree and the communication cost counts the number of messages exchanged at the *coordination level*.



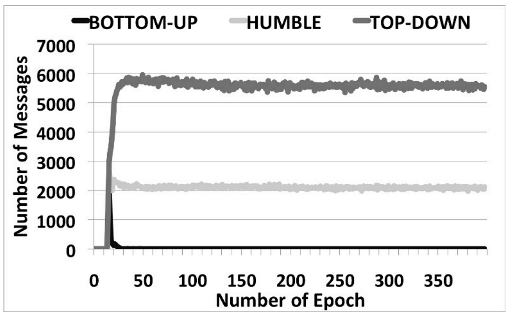
(a) Connectedness.



(b) Connectivity.



(c) Fitness.



(d) Communication cost.

FIG. 5.2. *Performance trade-offs for three adaptation strategies of AETOS:* BOTTOM-UP, HUMBLE *and* TOP-DOWN *[30].*

Figure 5.2a and 5.2b show that all three strategies build a well-connected tree topology within a few epochs. However, TOP-DOWN has the highest fitness of over 0.95 as shown in Figure 5.2c. HUMBLE and BOTTOM-UP follow with a fitness of 0.87 and 0.65 respectively. The high fitness of TOP-DOWN comes at a high communication cost of over 5000 messages per epoch as shown in Figure 5.2d. HUMBLE and BOTTOM-UP converge to 2000 and 0 messages per epoch respectively.

Figure 5.3 illustrates snapshots[7] of the tree topologies built by the adaptation strategies on the 350th epoch. The snapshots show visually the effect of high fitness in the tree topology built by TOP-DOWN.

These experimental results show that the ASMA architecture provides a high modularity and reconfigurability in the overlay service of AETOS. The performance of each architectural level can be separately studied and tuned, e.g., communication cost of the *coordination level* as shown in Figure 5.2d. A number of adaptation strategies provide different performance trade-offs. For example, in a network with limited bandwidth resources, BOTTOM-UP is the most effective, whereas, if network resources are not a constraint, a significantly higher fitness is achievable by TOP-DOWN.

The vertical interactions defined within the ASMA architecture can be used to manage and control these

[6]The adaptation strategies define the candidate parent and/or children with which each agent chooses to connect. BOTTOM-UP: the lowest ranked candidate parent; HUMBLE: the lowest ranked candidate parent and children; TOP-DOWN: the highest ranked candidate children.

[7]Visualisations are performed with the JUNG library [24] available at: `http://jung.sourceforge.net` (last accessed: December 2014)

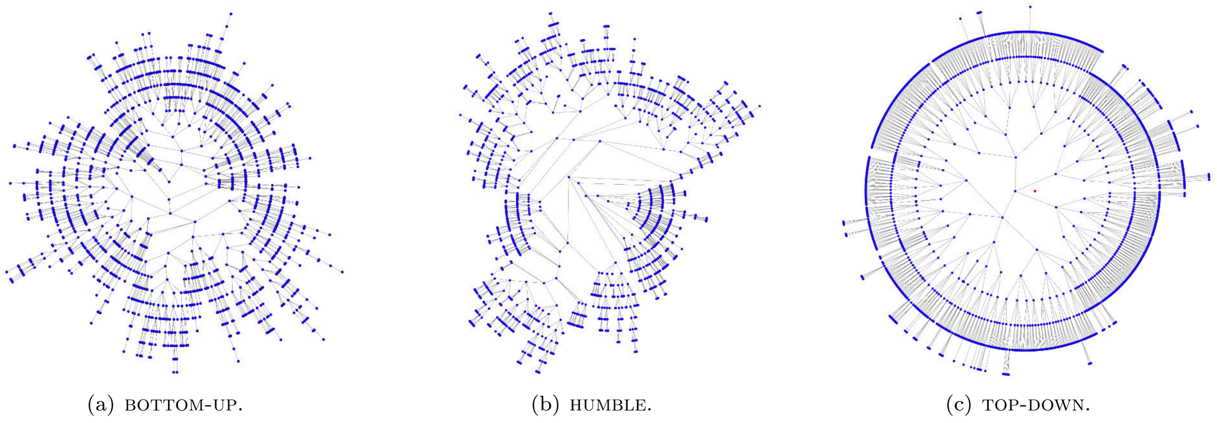


FIG. 5.3. *Visualisation of three adaptation strategies of AETOS on the 350th epoch [30].*

performance trade-offs. This section illustrates a scenario in which two adaptation strategies[8], PRESBYOPIC and BOTTOM-UP, are combined in such a way that a new strategy, HYBRID-03, inherits the high performance of each individual one without their limitations. The adaptation strategies are combined via a dynamic adoption scheme in which agents of the *structuring level* change from PRESBYOPIC to BOTTOM-UP during runtime. Switching can be performed automatically by monitoring the convergence of one of the performance metrics at the *coordination level*. If the monitored metric has converged, meaning its deviations are significantly lower than the ones during system startup, a switching signal can be included in the *structuring criteria*. This section shows the feasibility of designing hybrid strategies by empirically defining the switching time point on the 250th epoch and measuring the change of fitness and communication cost. Figure 5.4 illustrates the performance of HYBRID-03 under this scenario.

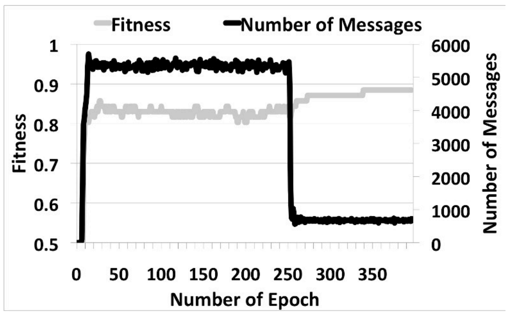


FIG. 5.4. *Perfromance of AETOS before and after switching from* PRESBYOPIC *to* BOTTOM-UP *on the 250th epoch [30].*

After switching from PRESBYOPIC to BOTTOM-UP, fitness increases 8% whereas, the number of messages drop from 5300 to 700 messages per epoch. Figure 5.5 illustrates the visualisation of the tree topology before and after switching strategies. Connectedness and connectivity are maximised. A higher fitness results in a higher number of nodes connected closer to the root of the tree after adopting BOTTOM-UP.

These performance enhancements are achieved without changing any architectural element of AETOS. Each layer realisation of ASMA retains its objective in each of the scenarios shown. ASMA provides a flexible, structured and modular architectural concept to realise and make manageable the complex system behaviour of AETOS.

[8]The agents choose to connect with the candidate parent and/or children as follows: PRESBYOPIC: the highest ranked candidate parent and the lowest ranked candidate children; BOTTOM-UP: the lowest ranked candidate parent.

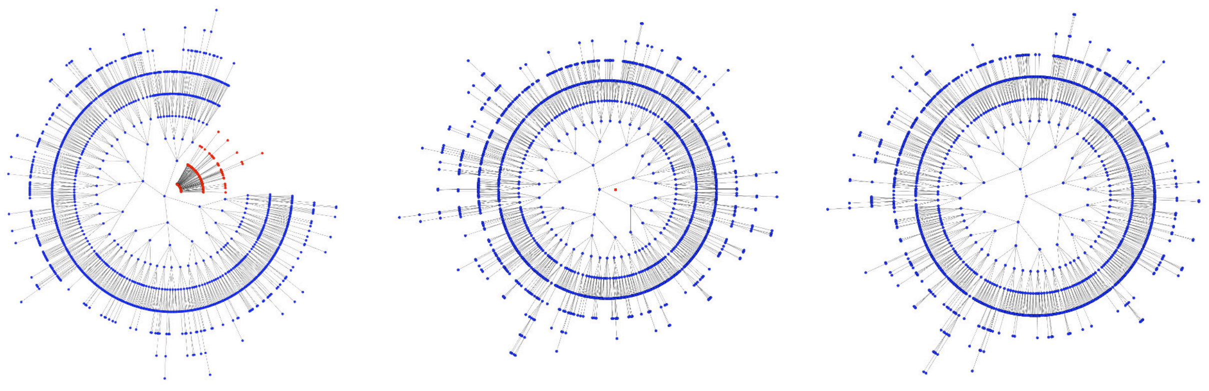

(a) 245th epoch. (b) 255th epoch. (c) 380th epoch.

FIG. 5.5. *Visualisation of* HYBRID-03 *before and after switching from* PRESBYOPIC *to* BOTTOM-UP *on the 250th epoch [30].*

5.2. DIAS: The Dynamic Intelligent Aggregation Service. DIAS performs decentralised aggregation by computing aggregation functions that receive as input states distributed in the peers of a network. A *state* represents a (aggregation) value of an application parameter at a specific point in time. A node may contain the *selected state* s'_i that is the one aggregated by nodes. During system runtime, the selected state s'_i may change and is equal to one and only one state from a finite number v of locally unique *possible states* $s'_i = s^0_i|s^1_i|...|s^{v-1}_i$. As the selected state changes, an earlier selected state is indicated as $\hat{s}_i$. Each node i contains an *aggregator* A_i, a *disseminator* D_i or both. An *aggregator* computes aggregation functions as $f(s'_0, s'_1, ..., s'_{n-1})$, whereas a *disseminator* provides the selected state to *aggregators*.

The *discovery level* of DIAS is realised by the gossiping protocol of the peer sampling service [14] as illustrated in Section 4.1. Gossiping guarantees that information is disseminated to all peers in a network to achieve accurate computations of aggregation functions. Agents disseminate and collect the IP address and port number of *aggregators* to which *disseminators* send their selected state.

The *structuring level* classifies *aggregator samples* received from the *discovery level* into three classes: (i) exploited, (ii) unexploited and (iii) outdated. The exploited *aggregators* of a *disseminator* D_i are the ones that have aggregated its earliest selected state s'_i. The unexploited *aggregators* of a *disseminator* D_i are the ones with which aggregation has not been performed and therefore, D_i has not provided any of its selected states to these *aggregators*. Finally, the outdated *aggregators* of a *disseminator* D_i are the ones that have aggregated a selected state of this *disseminator* earlier but since then the selected state has changed. Two adaptation strategies, EXPLOITATION and UPDATE, provide to the *coordination level* with priority either unexploited or outdated *aggregators* respectively. A third adaptation strategy, RANDOM, performs a random selection between unexploited and outdated.

Classification in these three classes is possible via local storage of *aggregation memberships* that is a representation of the information required to perform accurate aggregation. Accuracy indicates the quality of the DIAS overlay service and concerns the computation of aggregation functions without counting twice states or counting outdated states. An aggregation membership $M_{group}(member)$ of a certain 'member' to a certain 'group' is either positive or negative. Each agent of the *structuring level* in a peer i stores unique identifiers of possible states $\mathsf{S}^0_i, ..., \mathsf{S}^{v-1}_i$ corresponding to the actual possible states $s^0_i, ..., s^{v-1}_i$. Respectively, S'_i and $\hat{\mathsf{S}}_i$ refer to the unique identifiers of the selected s'_i and outdated $\hat{s}_i$ state in peer i. The *structuring level* stores a representation of the local states, their unique identifiers, and the *coordination level* stores the actual states, e.g., numerical or other type. The *structuring level* also uses the local unique peer identifier to map the local *aggregator* A_i and *disseminator* D_i. Therefore, $\mathsf{A}_i = \mathsf{D}_i$. An *aggregator* A_j and a *disseminator* D_i in two peers i and j perform horizontal interactions to exchange the required state information for aggregation. Four local aggregation memberships are involved in the aggregation performed:

MEMBERSHIP 1 ($M_{\mathsf{D}_i}(\mathsf{A}_j)$). *Membership of an aggregator in a disseminator.*

A *disseminator* D_i stores the identifier of an *aggregator* A_j to which it has disseminated its selected state at least once during an aggregation.

Membership 2 ($M_{\mathsf{S}_i^u}(\mathsf{A}_j)$). *Membership of an aggregator in a possible state.*

A *disseminator* D_i stores the identifier of an *aggregator* A_j for each possible state identified as S_i^u aggregated by this *aggregator*.

Membership 3 ($M_{\mathsf{A}_j}(\mathsf{D}_i)$). *Membership of a disseminator in an aggregator.*

An *aggregator* A_j stores the identifier of a *disseminator* D_i from which it has aggregated its selected state at least once during an aggregation.

Membership 4 ($M_{\mathsf{A}_j}(\mathsf{S}'_i)$). *Membership of a selected state in an aggregate.*

An *aggregator* A_j stores the identifier of a selected state S'_i aggregated from a *disseminator* D_i.

The *structuring level* efficiently stores these aggregation memberships in bloom filters. A *bloom filter* is a probabilistic data structure that efficiently stores membership information at a cost of false positives. The *structuring level* is able to detect and prevent inconsistencies in the aggregation originated from false positives by using mutual aggregation memberships between an *aggregator* and a *disseminator*, e.g., $M_{\mathsf{S}_i^u}(\mathsf{A}_j)$-$M_{\mathsf{A}_j}(\mathsf{D}_i)$ and $M_{\mathsf{S}_i^u}(\mathsf{A}_j)$-$M_{\mathsf{A}_j}(\mathsf{S}'_i)$ are mutual as they are representation of the same information.

Algorithm 17 and 18 illustrate the interactions of an *aggregator* A_i with a *disseminator* D_j. These interactions refer to the optional parts of the `adapt` and `consume` tasks in Algorithm 9 and 10 respectively. Aggregation is initiated by the `selectToProvide` subtask. Before a selected *aggregator* A_i is provided to the *coordination level*, a set of *outgoing criteria* is sent to A_i to make its aggregation memberships consistent to the memberships of *disseminator* D_j. Lines 1-7 of Algorithm 17 illustrate the update of the aggregation memberships by *aggregator* A_i. Update of the memberships is performed according the classification outcome of A_i by *disseminator* D_j. *Aggregator* A_i adds the memberships $M_{\mathsf{A}_i}(\mathsf{D}_j)$ and $M_{\mathsf{A}_i}(\mathsf{S}'_j)$ if it is classified as unexploited by D_j and additionally removes membership $M_{\mathsf{A}_i}(\hat{\mathsf{S}}_j)$ if it is classified as outdated. Aggregation is performed unidirectionally (flag='uni'), however, a bidirectional aggregation is performed if there is an *aggregator* A_j and a *disseminator* D_i (flag='uni-bi'). This option is checked in lines 9-18 of Algorithm 17.

Algorithm 17 Aggregation operations embedded in the `adapt` task of the *structuring level* in DIAS.

```
Require: incoming criteria: flag, class, $\mathsf{D}_j$, $\mathsf{S}'_j$, $\hat{\mathsf{S}}_j$
 1: add $M_{\mathsf{A}_i}(\mathsf{S}'_j)$
 2: if class=unexploited then
 3:     add $M_{\mathsf{A}_i}(\mathsf{D}_j)$
 4: end if
 5: if class=outdated then
 6:     remove $M_{\mathsf{A}_i}(\hat{\mathsf{S}}_j)$
 7: end if
 8: if flag='uni' then
 9:     if $M_{\mathsf{D}_i}(\mathsf{A}_j)$ : negative then
10:         outgoing criteria=getCriteria('bi', unexploited, $\mathsf{D}_i$, $\mathsf{S}'_i$, $\hat{\mathsf{S}}_i$)
11:         outgoing samples=getSamples('uni-bi', class, $\mathsf{A}_i$, outgoing criteria)
12:     else if $M_{\mathsf{D}_i}(\mathsf{A}_j)$ : positive and $M_{\mathsf{S}'_i}(\mathsf{A}_j)$ : negative then
13:         outgoing criteria=getCriteria('bi', outdated, $\mathsf{D}_i$, $\mathsf{S}'_i$, $\hat{\mathsf{S}}_i$)
14:         outgoing samples=getSamples('uni-bi', class, $\mathsf{A}_i$, outgoing criteria)
15:     else // $M_{\mathsf{D}_i}(\mathsf{A}_j)$ : positive and $M_{\mathsf{S}'_i}(\mathsf{A}_j)$ : positive
16:         outgoing samples=getSamples('uni', class, $\mathsf{A}_i$)
17:     end if
18:     return outgoing samples
19: else // flag='bi'
20:     outgoing samples=getSamples('bi', class, $\mathsf{A}_i$)
21:     return outgoing samples
22: end if
Ensure: outgoing samples
```

Similarly, the receipt of *incoming samples* by *disseminator* D_i triggers the update of its aggregation memberships as shown in lines 1-7 of Algorithm 18. *Disseminator* D_i adds the memberships $M_{\mathsf{D}_i}(\mathsf{A}_j)$ and $M_{\mathsf{S}'_i}(\mathsf{A}_j)$ if *aggregator* A_j is unexploited and additionally removes the membership $M_{\hat{\mathsf{S}}_i}(\mathsf{A}_j)$ if A_j is outdated. This completes a unidirectional aggregation. If a bidirectional aggregation is performed, the `adapt` task is executed with the *incoming criteria* as an input (line 9 of Algorithm 18).

The *coordination level* is responsible for the computation of aggregates. An aggregate is continuously computed based on an aggregation function provided by the aggregation *criteria*. The `parameterise` subtask of

Algorithm 18 Aggregation operations embedded in the `consume` task of the *structuring level* in DIAS.

```
Require: incoming samples: flag, class, $\mathsf{A}_j$, incoming criteria
 1: add $M_{\mathsf{S}'_i}(\mathsf{A}_j)$
 2: if class=unexploited then
 3:     add $M_{\mathsf{D}_i}(\mathsf{A}_j)$
 4: end if
 5: if class=outdated then
 6:     remove $M_{\hat{\mathsf{S}}_i}(\mathsf{A}_j)$
 7: end if
 8: if flag='uni-bi' then
 9:     adapt(incoming criteria)
10: end if
```

Algorithm 13 provides the aggregation function. Aggregates are updated by sending the value of the selected state to *aggregators* provided by the *structuring level* and classified as unexploited. If the provided *aggregators* are classified as outdated, the earlier selected state is sent as well. The `initialiseCoordination` subtask initiates this communication, the `finaliseCoordination` subtasks completes it and the `organise` subtask realises the computation of the aggregation functions as defined in Algorithm 13 and 14 of the ASMA architecture.

The *coordination level* forms an overlay network between *aggregators* and *disseminators* linked with overlay links that have two possible semantic values: unexploited or outdated but not exploited. Therefore, the aggregation functions computed exclude overlay links from the *coordination level* that result in duplicate aggregation values (exploited *aggregators*). The aggregation memberships, the classification process, the selections of *aggregators* are all complexity that is hidden from the aggregation process of the *coordination level*. Adaptation strategies tune the aggregation process in favour of (i) discovering new selected states in the system (EXPLOITATION strategy) or (ii) updating the aggregates with the most recent selected states (UPDATE strategy). The *coordination level* has to only provide the classification *criteria* that trigger this optimisation and inform about changes in the selected state. Therefore, the *coordination level* remains agnostic about the details of the optimisation. The aggregation *criteria* define how the aggregates are provided to applications, e.g., periodic delivery or delivery when aggregates converge to the actual aggregate values by monitoring a minimum deviation threshold.

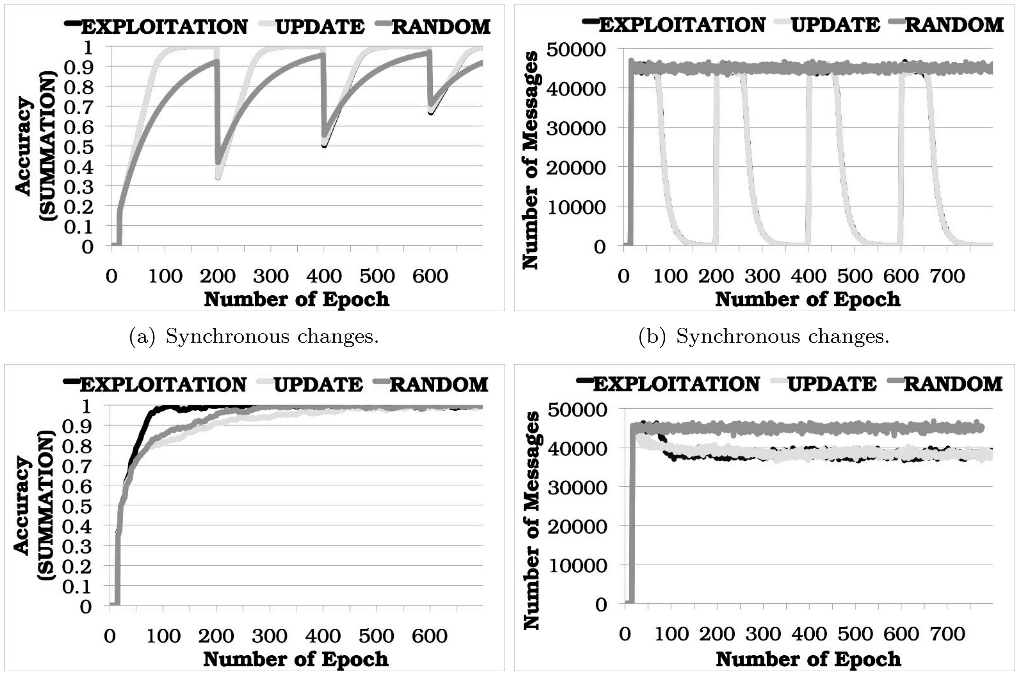
Figure 5.6 illustrates performance trade-offs between accuracy and communication cost for the adaptation strategies of DIAS. The accuracy measures how close the estimates of the aggregates, e.g., SUMMATION, is to the actual values [29]. Two scenarios are illustrated regarding how selected states change during runtime: (i) *synchronous* and (ii) *asynchronous* changes. In synchronous changes, the selected states of all peers in the network change simultaneously. In contrast, asynchronous changes occur arbitrary over time. In the illustrated experiments, synchronous changes occur every 200 epochs whereas in asynchronous changes, 420 selected states probabilistically change on average every 10 epochs.

Figure 5.6a and 5.6b show that after a synchronous change, accuracy drops dramatically and is restored to maximum within 100 epochs. This adaptation of aggregates causes a maximum of 45000 messages per epoch that drop to zero during the convergence period of 100 epochs. However, the communication cost of 45000 messages per epoch is constant for the RANDOM strategy.

Figure 5.6c and 5.6d show that despite the continuous changes of selected states every 10 epochs, DIAS is capable of maintaining a high accuracy after the initial convergence at system startup. However, a constant communication cost of 38000 messages per epoch is required to maintain this high accuracy that is yet lower than the 45000 messages per epoch of the RANDOM strategy.

DIAS can compute a wide range of aggregation functions at a performance comparable with the one shown in this section. In contrast to related methodologies reviewed in earlier work [29], DIAS does not require any architectural changes to compute a different aggregation function. It is because of the ASMA architectural modularity that the aggregation process is separated from routing. The two adaptation strategies provide a high reconfigurability in different networks settings. For example, during network scaling, EXPLOITATION is more effective than UPDATE. However, for a stable network with frequent changes, UPDATE is superior.

6. Comparison with Related Work. In earlier work [10, 11], the idea of 'open overlays' is introduced supported by a generic framework for overlay networks and their applications. This framework receives plug-



(a) Synchronous changes. (b) Synchronous changes.

(c) Asynchronous changes. (d) Asynchronous changes.

FIG. 5.6. *Performance trade-offs for the three adaptation strategies of DIAS:* EXPLOITATION, UPDATE *and* RANDOM *[29].*

in overlays defined by three components: 'Forwarding', 'state' and 'control'. These components are, in some aspects, similar to the three levels of ASMA. Applications are also introduced as plug-ins and are associated with possible overlay plug-ins that can support them. A plug-in can be positioned in the framework as independent or stacked with other plug-ins. Similarly with ASMA, top-down configurations are applied during deployment starting from the application plug-ins to the lower level network plug-ins. However, the number of possible combinations defined by the top-down configurations between the available overlay plug-ins can be large resulting in complex compositions.

Although dealing with overlay plug-ins is a generic, extensible and highly modular approach, the development of the three overlay components may be blended and cannot always be intuitive enough [10, 11]. Furthermore, the framework of overlay plug-ins does not define any high-level semantic of the component interactions. In contrast, ASMA provides a narrower defined context and objective for every self-organisation level. It also shows how these objectives are mutually supported via the exchange of *criteria* and *samples*.

Some related work [17, 20] focuses on supporting multiple overlays network capabilities as an IP-layer solution instead of a middleware solution that ASMA proposes for its realisations. OCALA [17] positions the 'overlay convergence' layer, built by an overlay-independent and an overlay-dependent component, under the transport layer. These layers provide a level of routing and lookup transparency between physical machines belonging to different overlay networks. However, there is a plethora of problems and open issues related to the support of existing IP-based applications, security, efficiency and access to overlay functions beyond routing. MOSAIC [20] is a declarative methodology for the composition of 'horizontal' (bridged via gateways) or 'vertical' (layered similarly to ASMA) overlay networks. Although this methodology provides a highly configurable and reasoning compositional environment for overlay networks, a large amount of information must be known a priori for each individual peer of the network. In addition, MOSAIC is highly dependent on a directory service that supports the composition process. It is unclear how changes to the directory service can be automatically reflected in the composed overlay networks. MOSAIC also faces the restrictions of an IP-layer solution similarly with OCALA. These approaches could in theory function complementary to ASMA overlay services for supporting communication between heterogeneous networks, e.g., wireless and wired networks.

iOverlay [18] provides an interface for building overlay networks and their applications. This interface is rather limited as it only supports overlay communication leaving excessive freedom to the developer. In comparison with ASMA, the main functionality of iOverlay corresponds to the *discovery level* of ASMA. A similar approach with iOverlay is followed by MACEDON [33]. Opus [4] is based on a backbone service to optimise the resource allocation for different applications. Therefore, the scope of this approach is limited compared to ASMA and the other approaches illustrated in this section. A multi-level economic framework for Grid services and resource allocation is earlier introduced [16]. Self-organisation is engaged for the discovery of agents that negotiate for resources. An overlay abstraction is provided to the agents. The system is designed based on web service technologies and therefore some of its components remain centralised. In contrast, ASMA introduces multiple self-organisation levels for system discovery, structuring and coordination without centralised components but in a collective fashion.

7. Conclusion and Future Work. This paper shows that collective adaptive systems can be designed and prototyped to provide modular and reconfigurable capabilities of a broad application scope: the overlay services. This paper contributes the ASMA conceptual architecture that guides realisations of complex overlay services via a few lines of high-level algorithmic expressions. The realisation of two overlay services according to ASMA together with the earlier experimental results illustrated in the new context of this paper empirically justify their higher abstraction, modularity and reconfigurability.

AETOS builds and maintains collectively different tree topologies with different topological properties that meet several application requirements. Topological reconfigurations in the self-organisation process are exclusively managed by plugged-in adaptation strategies that can be dynamically combined during runtime to improve performance under various scenarios such as node failures or network scaling. Similarly, DIAS computes almost any aggregation function that receives for input dynamically changing values distributed in a network. Adaptation strategies configure aggregation to compute in priority uncounted or outdated values depending on various network scenarios in which new nodes enter the network or regularly change their values. In both overlay services, the three levels of ASMA provide an intuitive and structured pathway to dissect the complex functionality of these systems in stand-alone, modular and reconfigurable subsystems. Although it is inevitable that this generic distributed computing approach has an impact on performance, e.g., high communication cost, overlay services allow reconfigurability with trade-offs. For example, the eight adaptation strategies of AETOS provide a spectrum of choices between high or low communication cost and performance. The dynamic adoption of adaptation strategies provide the option to explore this spectrum to improve the overall cost-effectiveness of overlay services.

Future work concerns the further realisation of overlay services according to ASMA. Usability case-studies and development scenarios shall strengthen the potential of a new distributed computing paradigm for collective adaptive systems based on overlay services.

Acknowledgements. The author would like to recall all people acknowledged in the preface of his PhD thesis [25] for making this research possible.

REFERENCES

[1] T. Anderson, L. Peterson, S. Shenker, and J. Turner, *Overcoming the Internet Impasse through Virtualization*, Computer, 38 (2005), pp. 34–41.

[2] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, *Model-Based Performance Prediction in Software Development: A Survey*, IEEE Transactions on Software Engineering, 30 (2004), pp. 295–310.

[3] A. Boulis, S. Ganeriwal, and M. B. Srivastava, *Aggregation in sensor networks: an energy-accuracy trade-off*, Ad Hoc Networks, 1 (2003), pp. 317–331.

[4] R. Braynard, D. Kostic, A. Rodriguez, J. Chase, and A. Vahdat, *Opus: an Overlay Peer Utility Service*, in Proceedings of Open Architectures and Network Programming, OPENARCH 2002, Los Alamitos, CA, USA, Jan. 2002, IEEE, pp. 167–178.

[5] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen, *Deployment Issues for the IP Multicast Service and Architecture*, IEEE Network, 14 (2000), pp. 78–88.

[6] J. Fan and M. H. Ammar, *Dynamic Topology Configuration in Service Overlay Networks: A Study of Reconfiguration Policies*, in Proceedings of the 25th International Conference on Computer Communications, INFOCOM 2006, Los Alamitos, CA, USA, Apr. 2006, IEEE, pp. 1–12.

[7] W. Galuba, K. Aberer, Z. Despotovic, and W. Kellerer, *ProtoPeer: A P2P Toolkit Bridging the Gap Between Simulation and Live Deployement*, in Proceedings of the Second International Conference on Simulation Tools and Techniques, ICST 2009, Gent, Belgium, Mar. 2009, ACM, pp. 1–9.

[8] C. Gkantsidis, M. Mihail, and S. Saberi, *Random walks in peer-to-peer networks: Algorithms and evaluation*, Performance Evaluation, 63 (2006), pp. 241–263.

[9] A. González-Beltrán, P. Milligan, and P. Sage, *Range queries over skip tree graphs*, Computer Communications, 31 (2008), pp. 358–374.

[10] P. Grace, G. Coulson, G. S. Blair, L. Mathy, W. Kit Yeung, W. Cai, D. A. Duce, and C. S. Cooper, *GRIDKIT: Pluggable Overlay Networks for Grid Computing.*, in Proceedings of Cooperative Information Systems, CoopIS 2004, vol. 3291 of Lecture Notes in Computer Science, Heidelberg, Oct. 2004, Springer-Verlag Berlin, pp. 1463–1481.

[11] P. Grace, D. Hughes, B. Porter, G. S. Blair, G. Coulson, and F. Taiani, *Experiences with Open Overlays: A Middleware Approach to Network Heterogeneity*, in Proceedings of the 3rd European Conference on Computer Systems, SIGOPS/EuroSys 2008, New York, USA, Apr. 2008, ACM, pp. 123–136.

[12] D. Haage, R. Holz, H. Niedermayer, and P. Laskov, *CLIO - A Cross-Layer Information Service for Overlay Network Optimization*, in Kommunikation in Verteilten Systemen (KiVS), Informatik aktuell, Heidelberg, Mar. 2009, Springer-Verlag Berlin, pp. 279–284.

[13] M. Jelasity, A. Montresor, and O. Babaoglu, *T-Man: Gossip-based fast overlay topology construction*, Computer Networks, 53 (2009), pp. 2321–2339.

[14] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, *Gossip-based Peer Sampling*, ACM Transactions on Computer Systems, 25 (2007).

[15] S. Jiang, L. Guo, and X. Zhang, *LightFlood: an Efficient Flooding Scheme for File Search in Unstructured Peer-to-Peer Systems*, in Proceedings of the 2003 International Conference on Parallel Processing, ICPP 2003, Los Alamitos, CA, USA, Oct. 2003, IEEE, pp. 627–635.

[16] L. Joita, O. F. Rana, P. Chacín, I. Chao, F. Freitag, L. Navarro, and O. Ardaiz, *Application deployment using catallactic Grid middleware*, in Proceedings of the 3rd International Workshop on Middleware for Grid Computing, MGC 2005, New York, USA, Nov. 2005, ACM Press, pp. 1–6.

[17] D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle, *OCALA: An Architecture for Supporting Legacy Applications over Overlays*, in Proceedings of the 3rd International Conference on Networked Systems Design & Implementation, NSDI 2006, Berkeley, CA, USA, May 2006, USENIX Association, p. 20.

[18] B. Li, J. Guo, and M. Wang, *iOverlay: A Lightweight Middleware Infrastructure for Overlay Application Implementations*, in Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware, Middleware 2004, vol. 3231 of Lecture Notes in Computer Science, New York, USA, Oct. 2004, Springer-Verlag New York, pp. 135–154.

[19] S. P. Mahambre, M. Kumar S.D., and U. Bellur, *A Taxonomy of QoS-Aware, Adaptive Event-Dissemination Middleware*, IEEE Internet Computing, 11 (2007), pp. 35–44.

[20] Y. Mao, B. T. Loo, Z. Ives, and J. M. Smith, *MOSAIC: Unified Declarative Platform for Dynamic Overlay Composition*, in Proceedings of the International Conference on Emerging Networking Experiments and Technologies, CONEXT 2008, New York, USA, Dec. 2008, ACM Press, pp. 1–12.

[21] R. Matei, A. Iamnitchi, and P. Foster, *Mapping the Gnutella network*, IEEE Internet Computing, 6 (2002), pp. 50–57.

[22] K. Nahrstedt, R. Chang, and C. Ward, *QoS-Assured Service Composition in Managed Service Overlay Networks*, in Proceedings of the 23rd International Conference on Distributed Computing Systems, ICDCS 2003, Los Alamitos, CA, USA, May 2003, IEEE, pp. 194–201.

[23] A. L. Nelson, G. J. Barlow, and L. Doitsidis, *Fitness functions in evolutionary robotics: A survey and analysis*, Robotics and Autonomous Systems, 57 (2009), pp. 345–370.

[24] J. O'Madadhain, D. Fisher, P. Smyth, S. White, and Y.-B. Boey, *Analysis and Visualization of Network Data using JUNG*, Journal of Statistical Software, 10 (2005), pp. 1–35.

[25] E. Pournaras, *Multi-level Reconfigurable Self-organization in Overlay Services*, PhD thesis, Delft University of Technology, March 2013.

[26] E. Pournaras, G. Exarchakos, and N. Antonopoulos, *Load-driven neighbourhood reconfiguration of Gnutella overlay*, Computer Communications, 31 (2008), pp. 3030–3039.

[27] E. Pournaras, M. Warnier, and F. M. T. Brazier, *Adaptation Strategies for Self-management of Tree Overlay Networks*, in Proceedings of the 11th IEEE/ACM International Conference on Grid Computing, Grid 2010, Los Alamitos, CA, USA, Oct. 2010, IEEE, pp. 401–409.

[28] E. Pournaras, M. Warnier, and F. M. T. Brazier, *Local Agent-based Self-stabilisation in Global Resource Utilisation*, International Journal of Autonomic Computing, 1 (2010), pp. 350 – 373.

[29] E. Pournaras, M. Warnier, and F. M. T. Brazier, *A generic and adaptive aggregation service for large-scale decentralized networks*, Complex Adaptive Systems Modeling, 1 (2013).

[30] E. Pournaras, M. Warnier, and F. M. T. Brazier, *Adaptive self-organization in distributed tree topologies*, International Journal of Distributed Systems and Technologies, 5 (2014).

[31] E. Pournaras, M. Warnier, and F. M. T. Brazier, *Peer-to-peer aggregation for dynamic adjustments in power demand*, Peer-to-Peer Networking and Applications, 8 (2014), pp. 189 – 202.

[32] M. Puviani, G. Cabri, and F. Zambonelli, *A taxonomy of architectural patterns for self-adaptive systems*, in Proceedings of the International C* Conference on Computer Science and Software Engineering, C3S2E '13, New York, NY, USA, 2013, ACM, pp. 77–85.

[33] A. Rodriguez, C. Killian, S. Bhat, D. Kostić, and A. Vahdat, *MACEDON: Methodology for Automatically Creating, Evaluating, and Designing Overlay Networks*, in Proceedings of the 1st International Conference on Networked Systems

Design and Implementation, NSDI 2004, Berkeley, CA, USA, Mar. 2004, USENIX Association, pp. 267–280.

[34] G. Tan, S. A. Jarvis, X. Chen, D. P. Spooner, and G. R. Nudd, *Performance Analysis and Improvement of Overlay Construction for Peer-to-Peer Live Media Streaming*, Simulation, 82 (2005), pp. 169–178.

[35] S. Tang, E. Jaho, I. Stavrakakis, I. Koukoutsidis, and P. V. Mieghem, *Modeling gossip-based content dissemination and search in distributed networking*, Computer Communications, 34 (2011), pp. 765–779.

[36] P. Van Mieghem, J. Omic, and R. Kooij, *Virus spread in networks*, IEEE/ACM Transactions on Networking, 17 (2009), pp. 1–14.

[37] R. Van Renesse, K. P. Birman, and W. Vogels, *Astrolabe: A Robust and Scalable Technology For Distributed System Monitoring, Management, and Data Mining*, ACM Transactions on Computer Systems, 21 (2003), pp. 164–206.

[38] J. Yuh-Jzer, F. Chien-Tse, and Y. Li-Wei, *Keyword Search in DHT-Based Peer-to-Peer Networks*, in Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, ICDCS 2005, Los Alamitos, CA, USA, Jan. 2005, IEEE, pp. 339–348.

[39] D. Zhenhai, Z. Zhi-Li, and Y. Hou, *Service Overlay Networks: SLAs, QoS and Bandwidth Provisioning*, IEEE/ACM Transactions on Networking, 11 (2003), pp. 870–883.

[40] H. Zhuge and L. Feng, *Distributed Suffix Tree Overlay for Peer-to-Peer Search*, IEEE Transactions on Knowledge and Data Engineering, 20 (2008), pp. 276–285.

[41] M. Wirsing, M. Hölzl, M. Tribastone, and F. Zambonelli, *ASCENS: Engineering Autonomic Service-Component Ensembles*, in Proceedings of the 10th International Symposium of Formal Methods for Components and Objects, FMCO 2011, (2013), Springer.

[42] M. Amoretti, *Introducing Artificial Evolution into Peer-to-Peer Networks with the Distributed Remodeling Framework*, Genetic Programming and Evolvable Machines, 2, (2013), pp 127–153, Springer.

[43] R. Bruni, A. Corradini, F. Gadducci, A. Lluch Lafuente, A. Vandin, *A Conceptual framework for adaptation*, in Proceedings of the 15th International Conference on the Fundamentals of Software Engineering, FASE 2012, Tallinn, Estonia, (2012), Springer.

[44] M. Amoretti, *Evolutionary strategies for ultra-large-scale autonomic systems*, Information Sciences, 274. (2014), pp. 1–16.

Edited by: Giacomo Cabri and Emma Hart
Received: Dec 15, 2014
Accepted: Jul 15, 2015

DOI 10.12694/scpe.v16i3.1101
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 16, Number 3, pp. 271–287. http://www.scpe.org © 2015 SCPE

SODAP: SELF-ORGANIZED TOPOLOGY PROTECTION FOR SUPERPEER P2P NETWORKS

PAUL L. SNYDER[*] AND GIUSEPPE VALETTO[†]

Abstract. Unstructured superpeer overlays are an approach to peer-to-peer systems that enables collective organization and ensures the efficient participation and use of divers peers with heterogenous capabilities and resources. Such overlays are, however, vulnerable to failures and attacks that target the superpeers in an attempt to disrupt the overlay. In this paper, we present *SODAP* (Self-Organized Degree Adaptation Protection), a self-organized strategy for the self-protection of the overlay, based on the local adaptation of a peer's degree in response to disconnections, whether directly detected, or indirectly discovered with the assistance of neighbor peers. When network conditions deteriorate, the SODAP mechanism induces the creation of redundant connections to superpeers, which leads to the adjustment of the entire overlay to a topology that is more resilient to disconnection, while still allowing the system to continue exploiting heterogeneous peer capabilities. When network conditions improve, SODAP responds by reducing peer degrees to reduce redundancy and streamline the topology. We demonstrate this mechanism and evaluate its effectiveness as an extension to *Myconet*, a self-organized superpeer overlay for unstructured peer-to-peer networks.

Key words: P2P, self-organization, self-protection, degree adaptation, topology adaptation

AMS subject classifications. 68M14

1. Introduction.

We present *SODAP* (Self-Organized Degree Adaptation Protection), a new self-protection strategy for peer-to-peer networks based on topology adaptation.

Peer-to-peer (P2P) networks are very large examples of highly decentralized collective systems that can scale to millions of participants. They are by nature open-boundary systems, where the individual peers are largely autonomous, often self-interested, and characterized by (even extreme) diversity in the capabilities and resources among participating peers. P2P networks must also adapt dynamically in the face of ever-changing conditions that may affect individual peers or large portions of the network. This requires decentralized control and self-organization, as central management or supervision is impractical or impossible.

P2P systems typically engage in collective self-organization through the construction of an overlay network; this imposes a topology on top of the often-chaotic underlying network, and acts as an enabler for other services (such as search or routing). Overlays are commonly classified as either structured (typified by Distributed Hash Table-based systems) or unstructured.

Unstructured overlays construct an arbitrary topology that can be optimized for the requirements of a particular application. A common unstructured P2P topology is a hierarchical overlay, based on the concept of *superpeer*. Superpeer overlays attempt to take advantage of peer heterogeneity, using protocols that identify particularly powerful or well-situated peers. Superpeers then differentiate their role and functionality, to improve the global performance and functioning of the overlay. Superpeers act as a backbone for the overlay, and provide services to less-powerful peers; they typically have a greater number of neighbors (hence, higher network degree) than normal peers, and may construct a scale-free topology [19].

Superpeer overlays are more resistent to random failures than overlays that are non-hierarchical [3]. However, they are more vulnerable to malicious attacks targeting specific peers [17]. The higher degree of the superpeers and their specialized role in the overlay increase the effect of these peers' failure; even relatively straightforward attacks—such as *crash attacks*—directed against these nodes may result in the disconnection of many other peers, and the disruption of the overlay at large, with relatively little effort on the part of the attacker [18].

Self-adaptation is an answer to this topological vulnerability. Several superpeer systems use *self-healing* protocols for the maintenance of the overlay that allow them to efficiently rebuild the intended topology after it has been damaged [19, 25]. Self-healing, however, remains purely reactive: it does not proactively *protect* the overlay, nor does it prevent its disruption for the duration of an attack.

[*]SunGard Consulting Services, New York City, NY, USA
[†]Fondazione Bruno Kessler, Trento, Italy

Another possibility is to augment the system with *self-protection.* One strategy, as suggested in [11] and [33], is *topology adaptation*, that is, adding another layer of adaptivity that dynamically modifies the rules of the protocol that constructs and maintains the overlay's topology, and in particular makes the choice of peer degree adaptive. By altering peer degree, the overlay topology can shift automatically to an overall degree distribution that is appropriate to the current conditions, and becomes more resilient to malicious attacks, as well as to variable levels of network churn.

A superpeer overlay with topology adaptation has two additional important traits, which, as observed by [12], are often key for a successful collective adaptive system. Not only does it exhibit *heterogeneity* and *temporal diversity*, but it has built-in means to leverage those characteristics for the benefit of the collective. The promotion from peer to superpeer is a mechanism for specialization, which instigates further heterogeneity through beneficial differentiation of roles, functionality and offered services among the system participants. Moreover, topology adaptation is a source of temporal diversity. Each individual peer can significantly change its behavior over time through how it "interprets" the rules of the the topology maintenance protocol, but these effects are not isolated. They also propagate to neighborhoods in the network, and, ultimately, to the whole overlay, which ends up exhibiting different, emergent topological properties at different times based on the circumstances.

Our SODAP topology adaptation protocol is an example of such collective adaptation for the purpose of self-protection. Under SODAP, peers locally adapt their neighbor connections in response to disconnections that are directly detected, or discovered collaboratively via neighbor signalling. We demonstrate and evaluate SODAP on top of Myconet, a self-healing, biologically inspired peer-to-peer overlay protocol [25]. One of the distinguishing characteristics of Myconet is that nodes are not simply divided between peers and superpeers, as in other P2P systems, but can specialize and evolve through a more refined hierarchy of roles, each with distinct functionality and responsibilities towards other peers and the overlay at large.

SODAP represents an advancement with respect to previous topology adaptation approaches, which tend to work in a "binary" mode, that is, make the entire network switch between two rigidly defined topologies. In contrast to those binary approaches, which—by design—choose to ignore peer heterogeneity when under attack, SODAP is able to adapt more smoothly and locally to both degradation and improvements in network conditions. As a consequence, SODAP achieves effective self-protection without giving up the ability to exploit peer heterogeneity.

The rest of this paper is organized as follows: in Section 2, we provide an overview of this field of research, and position SODAP in it. In Section 3 we discuss some technical details on our own previous work, that is, the *Myconet* and *HITAP* protocols that served as the inspiration for SODAP's topology adaptation mechanism, and how that previous work led to the design of the proposed protocol. Section 4 presents in depth the *SODAP* approach to topology self-protection through degree adaptation. Section 5 contains the results of our experimental evaluation of the protocol. Finally, in Section 6 we offer a recap of our results, discuss their significance, and outline some potential directions for future work.

2. Related Work. Peer-to-peer networks represent some of the largest and more widely used distributed systems; for example, content-sharing P2P networks drive very large percentages of Internet traffic [2]. Systems based on superpeer overlays have been used for file sharing (e.g. Gnutella, Kazaa), VOD streaming [21], VOIP (e.g. the original architcture of Skype [1]), and other heavy-duty, large-scale applications.

While P2P networks offer a number of significant benefits to the design and operation of open, complex, and large-scale distributed systems, they are faced with a raft of attacks that can be mounted against them. Surveys of these attacks can be found, for example in [5] and [6], which identify attack classes such as denial-of-service; man-in-the-middle, worms; rational attacks (that is, non-cooperation by network participants); sybil attacks (where nodes forge identities); and eclipse attacks (where colluding nodes attempt to partition the network). Yue *et al.* [31] identify additional classes of attacks, including those targeting the routing process, application-level attacks, and attacks that target the idiosyncratic topological features of a given P2P overlay. Of particular relevance to the self-protection mechanism discussed in this paper are *crash attacks* that aim to disconnect or seriously disrupt the fabric of a P2P network by disabling or eliminating a portion of the peers, such as superpeers, particularly those in topologically sensitive positions.

There are several classes of approaches in the P2P literature to counter those attacks. The conceptually simplest class is represented by non-adaptive *topology preservation* approaches, which aim to reinforce the network by building a fixed amount of redundancy into the fabric of the overlay. Yang and Garcia-Molina [3] suggest a simple method for increasing resilience using k-redundancy (discussing only the case where $k = 2$). In their approach, multiple superpeers share the duty of providing services to each cluster of leaf peers. The multiple-parent approach is taken in a different direction by ERASP [13], which proposes assigning a fixed number of superpeers for each leaf peer. THUP [28] examines a superpeer topology based on a bimodal degree distribution that is resistant to both churn and denial-of-service attacks, and suggests a join-strategy for peers that enter the network that allow them to approximate such a bimodal distribution. Different topology preservation strategies may be best suited for particular applications: for example, Brinkmeier *et al.* [4] examines topologies that are optimally stable against both peer churn and targeted denial-of-service attacks from the perspective of streaming video applications.

A step beyond building a fixed-topology redundant fabric is represented by adaptive *topology recovery* approaches that enable self-healing following damage to the overlay. This includes protocols such as Myconet, SG-1 [19] and DLPSPN [30], which can quickly heal the P2P overlay network after being disrupted by a successful targeted attack. This is a reactive defense that attempts to minimize the disruption to the overlay and any applications running on top of it.

Proactive and adaptive defense (*i.e.*, self-protection) is the goal of *topology adaptation* approaches. The key feaure of a self-protection strategy is its ability to respond immediately to critical disconnections, and rearrange the P2P fabric in ways that prevent its disintegration, and thwart attacks by rendering them ineffective or too costly. However, topology adaptation imposes itself a cost. as shown by [22], which examines the effect of failures of individual nodes on unstructured P2P networks, and analyzes the efficacy of increasing redundancy, degree of connectivity, and hop-count distance.

Self-protection via topology adaptation is a relatively open-field area of research. In the context of structured P2P overlays, [20] discusses how the Tapestry Distributed Hash Table [32] may identify and neutralize Denial-of-Service (DoS) attacks by modifying its topology in ways that isolate attacking peers in ad hoc clusters. For unstructured superpeer overlays, one of the primary previous works is Keyani *et al.* [11], which discusses the implementation of a binary self-protection strategy that switches a Gnutella overlay from a scale-free to an exponential topology. The switch occurs upon attack detection occuring in a relatively small proportion of the nodes (less than 15%). However, the initial scale-free topology is built using a centralized oracle, which does not provide support for reverting the topology after an attack is complete. Our own previous work, *HITAP* [26] (Hormone-Inspired Topology Adaptation Protection), similarly proposes a binary switch approach, but is fully decentralized and biologically inspired. HITAP uses the diffusion of an “alert hormone” through the nodes, to propagate the signal to switch between a superpeer and a flat topology, in response to targeted attacks against high-degree peers, and to switch back as the hormone level decays. HITAP proved effective in making a superpeer overlay substantially more impervious to those attacks. However, HITAP is limited, due to its binary protection strategy which cannot take advantage of peer heterogeneity during attacks; moreover, it is sensitive to the parameters of the diffusion protocol, which needs tuning to remain effective across multiple network scales, churn levels, and attack sizes.

In contrast with the binary, “all-or-nothing” stance of the works mentioned above, Zweig and Zimmerman [33] have offered a graph-theoretical argument that explores the feasibility and effectiveness of flattening progressively the range of node degrees in an overlay in response to attacks. Thart work has provided us with the inspiration for our SODAP project, which exhibits smoother topology adaptation capabilities. These work seamlessly and locally in both directions, increasing or decreasing the node degrees based on the observed disconnections (or lack thereof) of each node’s neighbors.

Finally, it is important to mention works that propose techniques to measure the topological vulnerability of P2P networks. Mitra *et al.* have examined this issue from the perspective of percolation theory [15, 18], measuring network stability as a critical fraction of nodes that must be removed before the network disintegrates, and proposing a framework for analyzing the resilience of P2P networks. This analysis is further advanced using rate equations to analyze the emergence of superpeers over time [16]. Percolation theory is also used by Srivastava *et al.* [27], who examine attacks on superpeer networks with varying levels of degree-degree correlation. Ghedini *et al.* [9] focus on the special problems caused by P2P systems characterized (like superpeer overlays) by a relatively small number of high-degree nodes. They suggest that considering only actual disconnections (or the use of standard degree and betweenness centrality measures to assess the imminent threat of disconnection) may not be enough to properly assess the state of a network. They suggest examining networks from the perspective of vulnerability as well as actual damage (the proportion of nodes that if killed would result in the disconnection of a network).

These techniques that assess topological vulnerability can be also useful to assess the performance of a given overlay protection strategy. However, they work by taking a snapshot of a topology and examining its instant resistance to attacks. Analysis is typically performed by progressively removing peers from the topology, either at random or through selection via a graph-theoretic metric. These approaches are not well-suited to evaluate the effect of adaptive protection strategies, such as SODAP, which continuously modify the overlay to thwart attacks. The experiments we present in Section 5 examine empirically similar properties as a dynamic feature of the overlay network, and show that SODAP effectively stabilizes the disconnection rate across a wide range of scenarios.

3. Background and Motivation.

Below, we discuss two relevant previous works of ours. Myconet (Section 3.1) is a biologically inspired superpeer overlay, which has especially efficient topology construction, maintenance and self-healing features, on top of which we have designed, developed and evaluated SODAP [25]. We also briefly describe (Section 3.2) HITAP, our previous topology adaptation work, since it provided the motivation and informed the design of SODAP. For full details, the interested reader should refer to [26].

3.1. Myconet.

Myconet is an unstructured P2P overlay based on a fungal metaphor. Superpeers are considered to be the *hyphae* (root structures) in a fungal mycelium, and non-superpeer leaf nodes are considered to be *biomass* that can be moved among the hyphae as needed. Myconet uses an abstract concept of *capacity* to model the heterogeneity of peer capabilities: higher-capacity nodes make better superpeers, because they are able to maintain connection to (and provide application-specific service to) more of the other peers. To preserve the generality of Myconet, the mapping of capability and resource profiles to this abstract and uniform capacity measure is considered to be an application-dependent exercise.

In Myconet, local protocol dynamics work in a decentralized and self-organized fashion to identify the "best" participants to the overlay (according to the capacity metric) to serve as superpeers from a heterogeneous assortment of peers. In Myconet, superpeers (a.k.a *hyphal peers*) evolve through a series of protocol states, as displayed in Figure 3.1. Each state corresponds to a distinct role, and the peers collaborate, each according to its current role, to build a robust network of interconnections with other hyphal peers. They also perform a self-healing function, as each takes local actions role to reconstruct and maintain the overlay in response to incidental peer failures or crash attacks. When promoted from biomass to the first, *extending* hyphal state, superpeers are mainly focused on exploration, acting as connection points for new biomass peers entering the network. *Branching* hyphae are extending superpeers that have reached their target level of utilization (*i.e.*, they have connected to sufficient biomass peers to reach a level of utilization appropriate to their capacity); they manage the number of extending peers in the network while growing new hyphal interconnections. Finally, *immobile* hyphae have reached levels of full biomass connection and a target number of hyphal interconnections, and are considered to be relatively stable, having remained in the network long enough to transition through all the previous protocol states. When hyphal peers contact each other, superpeers of lower protocol state and lower capacity may transfer a portion of their attached biomass peers towards higher superpeers, in order to saturate the capacity of the latter. Should a hyphal peer for any reason fall below a utilization threshold, it may demote itself to a lower protocol state, possibly reverting to become a biomass peer.

Myconet constructs and maintains a strongly interconnected, superpeer-based overlay that converges quickly to an optimal number of superpeers. It self-heals equally quickly in the face of failures, repairing damage and dynamically adjusting the network topology in response to changing conditions. We have applied Myconet as the overlay substrate for a decentralized service network with clustering and load-balancing, described in [29].

3.2. HITAP.

HITAP (Hormone-Inspired Topology Adaptation Protection) is a biologically inspired approach to self-protection of a P2P network, switching between two different topology management protocols in response to detected attacks. HITAP has been built on top of the basic Myconet protocol, though its self-protection strategy is also applicable to other unstructured superpeer-based systems.

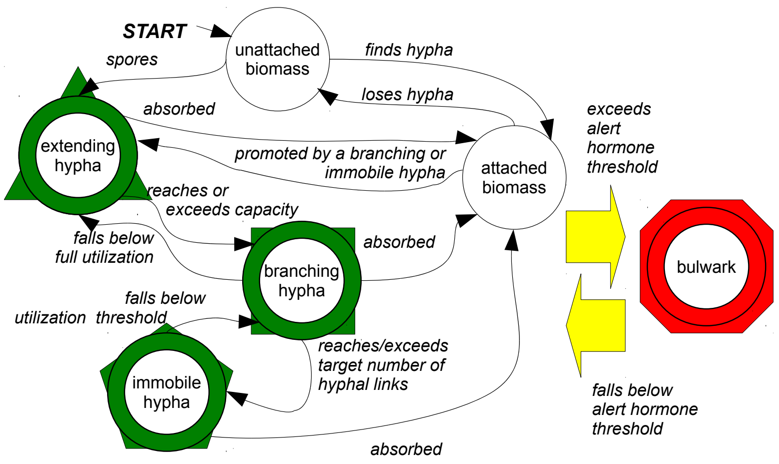

START
unattached biomass
spores
finds hypha
loses hypha
absorbed
extending hypha
promoted by a branching or immobile hypha
attached biomass
exceeds alert hormone threshold
reaches or exceeds capacity
falls below full utilization
absorbed
branching hypha
bulwark
falls below utilization threshold
reaches/exceeds target number of hyphal links
immobile hypha
falls below alert hormone threshold
absorbed

Fig. 3.1. *Protocol state transitions in Myconet (left) with the added bulwark state used by HITAP (see Section 3.2). States are are described in Section 3.*

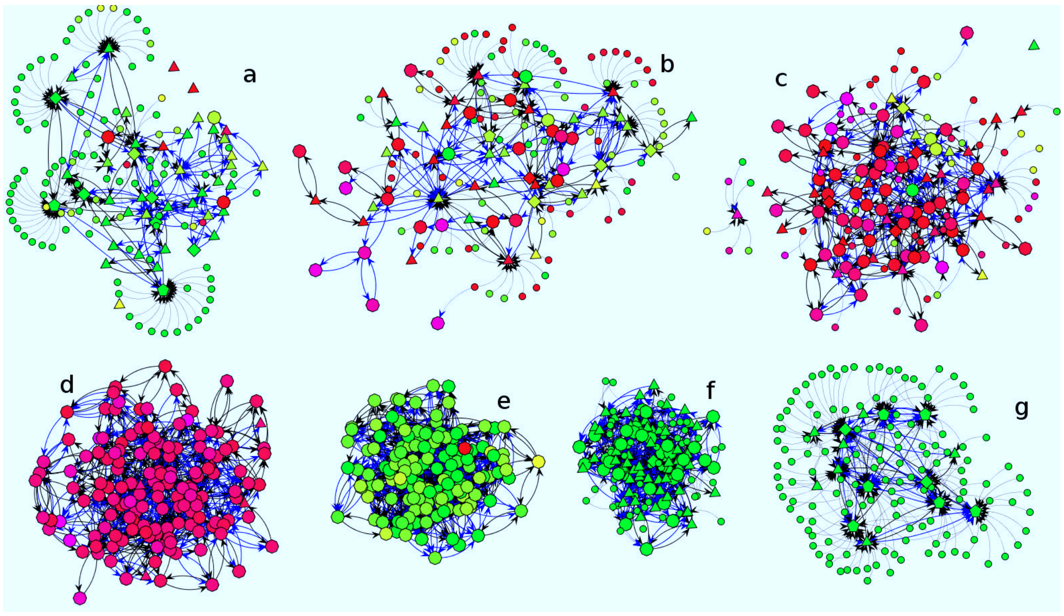


Fig. 3.2. *HITAP attack detection in an example 150-node network with* $c_{max} = 20$ *(see Section 5.1 for a description of this parameter). (a) Two cycles into an attack, some local buildup of alert hormone is occuring. (b) Four cycles into the attack, nodes have passed the transition threshold (red) and switched into bulwark (octagons), causing more nodes to make the jump. (c) Alert hormone levels continue to rise, diffusing through all nodes. (d) Six cycles into the attack, all nodes have switched to the bulwark state. (e) Four rounds after the attack ends, hormone concentrations in most nodes have decayed below the reversion threshold; nodes delay their return by five cycles to reduce thrashing. (f) Nine cycles after the attack ends, alert hormone levels have decayed and the nodes revert to normal operation. (g) Within six more cycles, the nodes have reconstructed the superpeer overlay.*

Damage to the overlay caused by peer disconnection (whether by attack or churn) results in HITAP in the generation of a marker (metaphorically considered to be an *alert hormone*) that spreads from neighbor to neighbor by diffusion. The increasing concentration of the hormone in a system experiencing a targeted attack causes the network as a whole to adapt its topology and switch to a “flat” mode where all nodes have a nearly uniform degree. While the system remains in that mode, attackers are deprived of major targets and the network becomes much harder to disrupt. As the attack subsides (or is reduced to simply killing random low-degree nodes), the hormone levels of each node decay. When the local concentration falls below a threshold, nodes will switch their protocol back to normal operation, and start working towards re-establishing a superpeer overlay.

The exact amount of hormone generated by the remaining peers is determined by the number of former neighbors of a failed peer, according to a quadratic relationship. This function places a heavier weight on the failure of high-degree peers (such as those in the immobile or branching states), whereas it is relatively insensitive to the failure of low-degree peers (such as biomass and peripheral extending hyphal peers). Since every neighbor of a failed peer will generate the alert hormone, the failure of large-degree peers will result in the release of large amounts of hormone throughout the network.

Once sufficient amounts of alert hormone have accumulated in a peer, it switches into a new protocol state, termed *bulwark*. Bulwark peers have switched into a defensive posture and act to reduce the network's vulnerability to attack. They do not follow the full Myconet protocol, but simply work to reconfigure their local neighbor relationships into a flat (and hence non-superpeer) network by maintaining a fixed number of connections to other peers. When a peer switches into the bulwark state, it severs connections with all its existing neighbors, causing the cascading generation of yet more alert hormone.

The maximum quantity of alert hormone that may be held at a node is capped, to prevent the build-up of network pockets with very large amounts of hormone. Additionally, peers metabolize the alert hormone over time. The local concentration is periodically reduced by a fixed percentage. Thus, hormone levels will spike following a failure, but will slowly drop back down if additional failures do not continue to generate additional alert signals.

After switching states, a new bulwark peer immediately begins to connect to new neighbors, growing new connections to other peers until it has reached a target range (as well as dropping connections that bring it above that range). If all peers enter the bulwark state, the network will quickly converge to a relatively flat topology, without superpeers. The network thus becomes resilient against targeted attacks, although, while in this mode, the network cannot effectively exploit heterogeneous node capabilities.

Once the network has shifted to such a flat configuration, attacks against particular peers are practically indistinguishable from ordinary churn. Because of this, the failure of a bulwark peer does not cause significant levels of the alert hormone to be generated.

A peer will remain in the bulwark state until its local concentration of alert hormone has dropped below a threshold value, plus an additional period of time. This latency is a customizable parameter, and is designed into the protocol to prevent the network from attempting to switch between modes too quickly, which may result in the overhead of restoring the superpeer topology while waves of hormones are still traversing the network, possibly because an attacker is still trying to disrupt the overlay.

3.3. Motivation. HITAP has demonstrated a decentralized, self-organizing approach to self-protection that is able to detect and mitigate topology-based attacks.

However, HITAP also has a relatively coarse-grained mechanism: the efficiency of the systen is greatly reduced while the network is in the defensive posture, as the overlay is no longer taking advantage of peer heterogeneity. A further challenge is the sensitivity of the hormone-based signaling mechanism used by HITAP to network conditions. Values selected for the protocol parameters—in particular, values that regulate the generation and decay of the alert hormone—have a significant effect on the system's ability to distinguish between attack conditions and normal operations, in which some background level of peer churn is natural; however, there is no single parameter set that is equally effective across a wide combination of churn rates and network sizes. Finally, HITAP tends to fall victim of its own success: when the overlay has assumed the defensive stance of the flat topology, attacks become ineffective but also indistinguishable from background noise and random failures. Therefore, HITAP is not able to detect with certainty that an attack has finished; rather, it "guesses" when to begin rebuilding the superpeer network, based on the decay rate of the alert hormone (another parameter). If the hormone evaporates too quickly, the attack may still be underway, and the network will need to again revert to the bulwark mode, resulting in further inefficiency.

These limitations have prompted us to investigate alternate strategies for overlay self-protection through topology adaptation. The outcome is *SODAP*, where nodes adjust their degrees dynamically to improve resistance to both attacks and network churn while continuing to exploit high-capacity peers as superpeers.

4. Approach. We now describe *SODAP* (Self-Organized Degree Adaptation Protection) as we have implemented it on top of the basic Myconet protocol; however, it is important to underline that its approach is sufficiently general to be applicable also to other superpeer-based P2P overlays. While the inspiration for SODAP's design came from our experimental work developing HITAP, it is based on a quite different approach. Rather than attempting to determine whether an attack is underway, SODAP continually and dynamically adjusts the number of connections to parent superpeers held by leaf peers, based upon disconnection events locally detected by each peer. This has the additional benefit of allowing the network to self-optimize to avoid disconnections due to peer churn.

Algorithm 1 SODAP Superpeer Maintenance Mechanism

```
$t_n \leftarrow 1$
loop
    for $i \leftarrow 1 \ldots hops_{max}$ do
        $fail_i \leftarrow \max_i$[failuresReceivedOfSize($i$)]
        if $i < hops_{max}$ then
            announceDisconnection($fail_i, i+1$)
        end if
    end for
    $f_{max} \leftarrow \max_i[fail_i]$
    if nodeIsSuperpeer then
        if wasDisconnected then
            $t_n \leftarrow t_n + 1$
        else if $f_{max} > t_n$ then
            $t_n \leftarrow t_n + 1$
        else if rand $< Prob_{decay}(t_n)$ then
            $t_n \leftarrow t_n - 1$
        end if
        if neighborCount $< t_n$ then
            connectToNewSuperpeer
            if $CallwasDisconnected$ then State announceDisconnection($t_n - 1, 0$)
            end if
        else if neighborCount $> t_n$ then
            disconnectFromRandomSuperpeer
        end if
    end if
end loop
```

Mechanism. The process followed by each SODAP peer is shown in pseudocode in Algorithm 1. A peer P_n keeps a *parent target* (t_n), which is the number of superpeers to which that peer will attempt to maintain connections, whenever it finds itself in the leaf state (*i.e.*, Myconet's *biomass* state). The initial value of t_n is 1, and in perfect network conditions (where no abrupt disconnections are observed) will remain there.

Similar to the mechanism underlying HITAP, the disconnection signal is propagated by the peer that receives it to its neighbors. The SODAP signal is simpler than the hormonal diffusion of HITAP, propagating for a fixed range ($hops_{max}$, the maximum number of hops). Peers receiving this signal track the highest value $fail_i$ observed within a time window for each hop distance $1 \leq i \leq hops_{max}$, as well as the highest value f_{max} observed overall. $hops_{max}$ is a SODAP protocol parameter; through experimentation, we have observed that a value of 2 is sufficient across for a wide range of scenarios in a Myconet overlay.

When a peer detects that it has been disconnected due to the unexpected failure of a neighbor peer, it responds by incrementing its individual t_n value by one. By creating additional links to multiple superpeers, the chance of the node becoming disconnected due to future failures is reduced. Also, since a superpeer's capacity dictates how many leaf peers it can service, after a significant proportion of leaf peers have raised their targets, new superpeers will be dynamically recruited by the regular topology maintenance protocol of Myconet, in order to handle the additional leaf peers connections.

Any peer receiving this failure announcement may also adjust its parent target in response. If the largest failure announcement a peer P_n receives within a given time window is f_{max}, then P_n will increment its own parent target t_n by one if $f_{max} > t_n$.

Similarly, if the largest failure observed is less than P_n's parent target, P_n may adjust its parent target downward by applying a decay probability function, in an attempt to reduce unnecessary redundancy and improve efficiency. However, that observed failure must be two less than P_n's current target ($f_{max} < t_n + 1$) before P_n will consider taking this action, since if f is only one less than t_n, a reduction of t_n would place the new target at a "danger" level where a failure was actually observed.

The decay function used determines the probabiilty that P_n will decrement its parent target t_n by one. The function used by SODAP is:

$$Prob_{decay}(t_n) = \frac{1}{1 + e^{(t_n/x)-s}}$$

This is a sigmoid function that provides a relatively small probability of decay if the current parent target is low but rapidly increases that probability as that parent target increases. Experiments have demonstrated that values of $x = 2.5$ and $s = 3$ work well across all tested scenarios.

P_n will make no adjustment to its parent target t_n if the largest failure announcement f_{max} is equal to or one less than its current target.

If a new peer P_c were always to enter the network with an initial parent target of $t_c = 1$, its chances of disconnection might be relatively high if network conditions are poor. In order to take advantage of current peers' knowledge of current network conditions, a peer that is entering the network and is connecting to a superpeer for the first time will set its parent target to mirror the target of that superpeer. Thus, if P_c connects to a superpeer S with a parent target $t_s = k$, P_c will set its own initial parent target $t_c = k$.

After being disconnected and raising its parent target, a peer P_n then attempts to connect to a new superpeer. It then sends a signal to that superpeer that it is making a reconnection, and also communicates the t_n level to which it was operating when the disconnection occurred. Thus, when a peer with $t_n = 2$ is disconnected by failure, it raises its target to $t'_n = 3$ and, after connecting to a superpeer S, announces to S that operating at $t_n = 2$ was insufficient to prevent disconnection. Following connection to S, P_n will continue—if necessary—to make connections to additional superpeers, until it has reached its t_n target.

SODAP Implementation in Myconet. In order to accommodate the self-protection mechanism of SODAP within Myconet, we had to make only simple modifications to the implementation of the Myconet topology maintenance protocol. We added the functionality for failure detection, failure announcements, and maintenance of parent targets as discussed above; moreover, the behavior of peers in certain Myconet protocol states was adjusted, specifically peers assuming the *biomass* (leaf) role, and the *extending hyphal* role.

When a peer P_n has a parent target of $t_n = 1$, it will operate using the normal Myconet rules. Whenever $t_n > 1$ and P_n is in the *biomass* state, it will attempt to connect to multiple superpeers (rather than the single parent of the basic protocol). If a biomass peer has excess superpeer connections above its t_n target, it will instead drop those connections.

This simple local behavior of biomass peers has global consequences: it will gradually cause the increase (or decrease) of the number of hyphal nodes in the network, since the same peer will now be counted as a client multiple times, by different hyphae. No modifications to the hyphal rules are required in order to achieve this scaling behavior.

One further modification is required for superpeers in the *extending* state. Normally, these peers maintain at least one connection to a higher-state (*branching* or *immobile*) superpeer. However, in network conditions where parent targets greater than 1 are needed to maintain network connectivity, this behavior would make *extending* peers into a topological weak point. This is addressed by having the *extending* nodes maintain a number of connections to higher-state superpeers equal to their current parent target.

A final adjustment to the Myconet rules has also been made in order to improve the distribution of the failure alerts through the network. Under normal rules, only *extending* peers will accept connections from new *biomass* nodes entering the network, since higher-state superpeers are typically at or near their desired utilization levels (pulling leaf peers from lower-state superpeers as needed in order to maintain this level). This behavior would cause failure announcements to be concentrated where *extending* peers are connected. Since the number of extending peers in stable networks may be quite low, those announcements would originate only in few locations in the network. Therefore, we allow all peers to accept connections up to 105% of their target number of clients; in this way, the chance that each peer is likely to see an announcement which is reflective of the current state of the network is much improved.

5. Evaluation. As mentioned in Section 2, the research landscape on the self-protection of superpeer overlays via topology adaptation is relatively sparse. Because of that, a benchmark for comparing and contrasting different topology adaptation approaches does not yet exist. For our evaluation, we report in quantitative ways the self-protection benefit of SODAP by comparing it to the same attack on a similar overlay without the topology adaptation mechanism. This allows an assessment of the level of support and the strength of the defense offered by our self-protection protocol.

We have implemented SODAP on top of Myconet, using an existing, cycle-based simulation (described in detail in [25] and [29]) based on the PeerSim framework [10]. PeerSim is a Java-based platform that has been used extensively in the P2P research literature for evaluating a wide variety of peer-to-peer protocols.[1] Simulation is the premier experimental approach for testing, validating and evaluating P2P protocols at scale, due to the difficulty and expense of building, deploying and testing systems with tens of thousands of nodes (or orders of magnitude more) in the lab or in the field [23]. The behavior of the underlying Myconet toplogy maintenance protocol has also been validated in a live distributed environment at smaller scales [14] with an implementation built using the Protopeer framework [8].

5.1. Experiment Design. The Myconet protocol has a number of parameters, and we have chosen their settings based on values used in our previous work for ease of comparison [25, 26, 29]). The C_n parameter controls the target number of inter-hyphal links that branching and immobile nodes maintain, and influences the level of resilience of the superpeer Myconet overlay to disruptive events. For all experiments, we used $C_n = 5$; a value that has been validated empirically in our previous work as providing a desirable balance between resilience and efficiency at very diverse scales. Myconet peer capacities are assigned using a power-law distribution (capped at a maximum value c_{max}) such that the probability of a node having capacity x (with x in the range $[1 \leq x \leq c_{max}]$) is $Prob\,[c_n = x] = x^{-\alpha}$. We chose $c_{max} = 500$ for experiments with networks of 10,000 peers, as this value is used for evaluating similarly scaled protocols in [19] and [13]. Since very large values of c_{max} relative to the number of peers in the network tend to produce degenerate superpeer topologies (with most nodes clustering around a very few nodes with extremely high relative capacity), we reduced c_{max} to 50 for the 1,000-peer experiments.

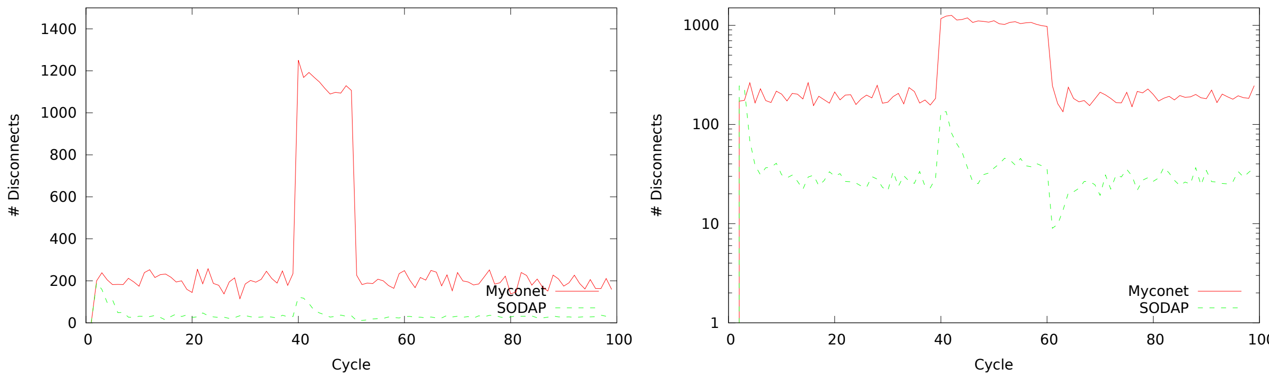
In our experiments, we simulate denial-of-service attacks against the most important peers in the network as used in our previous work [26] (and similar to the attack described in [33]). During this attack, the k highest-degree peers are removed each cycle. Attacks begin at round 40 (after the superpeer topology has been constructed by the Myconet protocol and has stabilized, with easily recognizable, prominent hyphal peers) and continue for either ten or twenty cycles. After these peers are removed, k new peers are added to the network. These peers' capacities are drawn from the same power-law distribution used for the original assignments; this may result in a decrease in the capacity of the most powerful peers over time, as the new peers added are unlikely to be as large.

Churn (the rate at which peers join and leave the network) is a significant factor in the performance of peer-to-peer networks. As SODAP does not explicitly consider the age of peers in its dynamics, the experiments discussed in this section use a simple model of churn, where a fixed percentage of nodes are removed each round, and a similar number are added. While sophisticated models of churn have been examined in the research literature (as discussed in Section 2), SODAP's performance is expected to remain consistent across differing churn models where similar percentages of peers enter and and exit the network within a similar time period. As with the attack scenarios, the peers added during churn may have differing capacities from the peers that were removed.

Our primary metric for the SODAP protocol is the disconnection rate; that is, the number of peers that each cycle lose their connections to the rest of the overlay as the result of another peer's exiting the network, whether from churn or an attack. New peers entering the network for the first time (following an attack or churn operation) are not counted towards this disconnection rate until they have successfully connected to one other peer at least once.

We also use metrics for the average degree of the superpeers (hyphae) and the number of superpeers in the network overall. Where we show the *optimal* number of superpeers in our charts, that number has been

[1]A partial list of the many protocols that have been implemented using PeerSim (including links to publications) can be found at `http://peersim.sf.net/#code`.



(a) 10-round attack (linear y-scale)

(b) 20-round attack (log y-scale)

FIG. 5.1. *Disconnection rates for reference experiments (10,000 nodes, 2% churn, attack beginning at cycle 40)*

calculated offline using an oracle: the largest peers in the network are progressively selected until the total capacity of these selected peers is sufficient to service all of the remaining peers in the network.

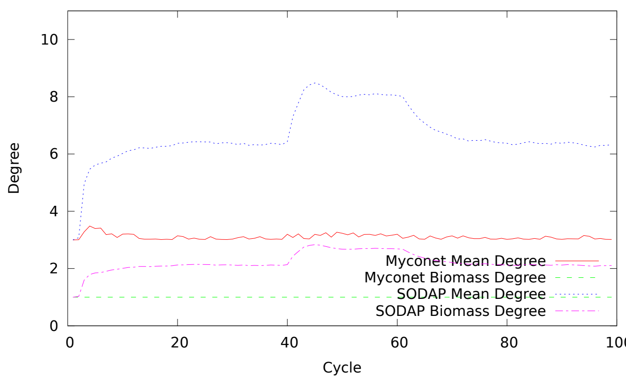
Each experiment discussed in this section consists of 100 independent runs, each lasting 100 cycles; we report the average of a set of metrics over those 100 runs.

5.2. Reference Experiment. We have selected a reference experiment configuration as a baseline for evaluation and discussion, and present the results for that configuration in this subsection. Experimental results discussed in the other subsections of this section vary one parameter of this configuration to facilitate direct comparison of results while examining the performance of SODAP under a range of conditions. The same experiments were also run with the basic Myconet protocol, and results are compared to show the improvements provided by dynamic degree adaptation. Although Myconet was used for expediency and consistency, in these comparisons it is simply a representative superpeer protocol for the maintenance and construction of unstructured superpeer overlays, and we maintain that the results presented hereby can extend to other protocols.

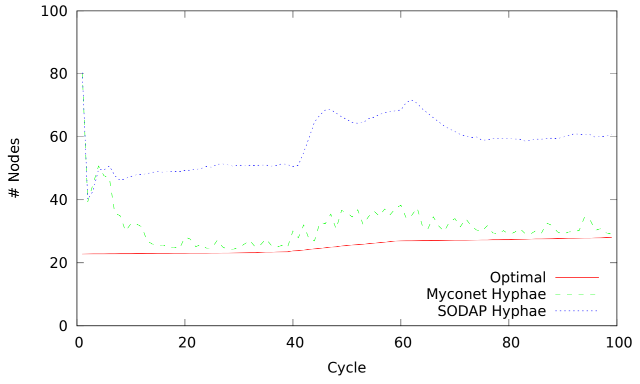
For our reference experiment, we use a network with 10,000 peers, $c_{max} = 500$, $C_n = 5$, and 2% churn per cycle. A single attack is started at round 40 with $k = 2$ (removing the two largest-degree nodes each cycle). Attack lengths of both 10 cycles and 20 cycles were evaluated. Often results for 10 cycles and 20 cycle attacks are similar, so both results are not shown for all experiments.

As can be seen in Figure 5.1, during normal network operations with continuous peer churn, SODAP provides a major improvement in the disconnection rate when compared to the basic Myconet protocol, resulting in a reduction of around 90% in the disconnection rate. As can be seen at the start of the experimental run, both basic Myconet and Myconet with SODAP begin with around 200 peers being disconnected as a result of the 2% churn. In response to these detected failures, SODAP then locally increases the parent target of biomass peers, reinforcing the overlay. These effects are shown in Figure 5.2.a, which shows the adaptative response of SODAP: Myconet node degrees do not change, while SODAP degrees quickly increase. The mean degree of biomass peers increases (purple dashed-dotted line) with the increased number of parents, and the overall mean degree of peers in the network increases due the biomass peers as well as to the increased number of of peers that specialize into a superpeer role (blue dotted line). The additional superpeers that promote themselves to handle the additional connections are shown in Figure 5.2.b. This added robustness results in a decrease in the efficiency of the topology: during normal 2% churn, the number of superpeers stays at around 2.5 times the number that would be needed in optimal conditions.

It is during a targeted attack (rounds 40–60) that SODAP shows its merit most clearly. Without degree adaptation, in the basic Myconet network around 11% of all peers are severed each round. Although Myconet's self-healing capabilities may repair the overlay efficiently, the system remains effectively disabled until the attack subsides. SODAP, in comparison, responds quickly as the attack is detected, creating redundant connections that reduce the disconnection rate to one only slightly higher than during normal churn (Figure 5.1).

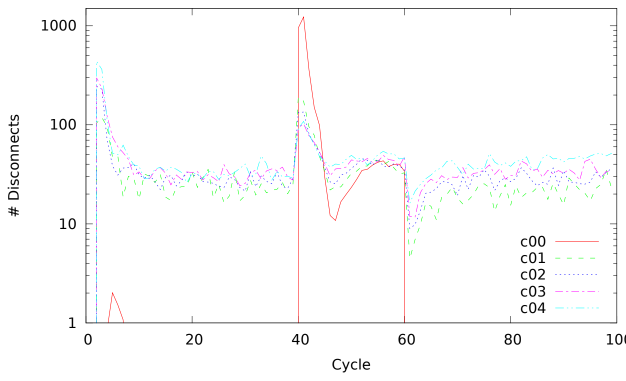


(a) Mean degree (all peers and biomass peers)

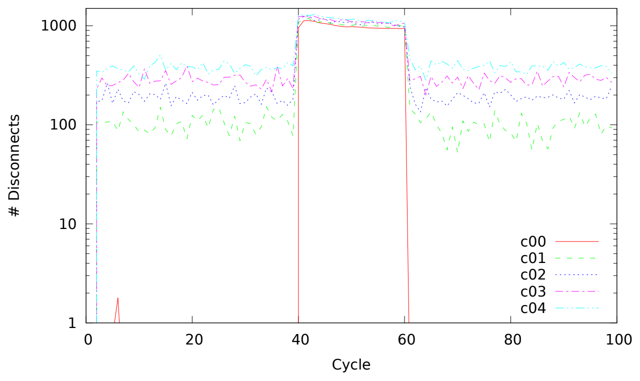


(b) Superpeer counts

FIG. 5.2. *Reference experiments (10,000 nodes, 2% churn, 20-round attack beginning at cycle 40)*



(a) SODAP disconnection rate



(b) Myconet disconnection rate

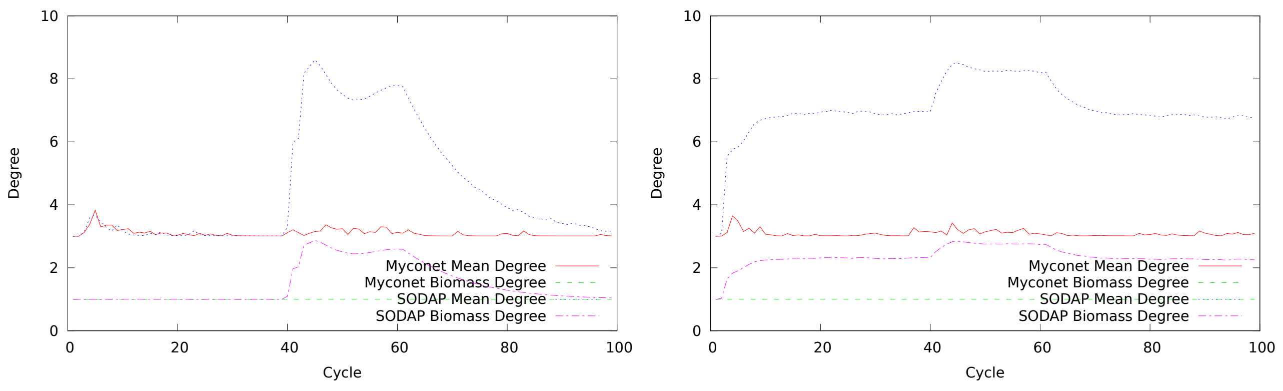
FIG. 5.3. *Disconnection rates varying churn rates (10,000 peers, 20-round attack)*

Following the attack, peers begin to lower their parent targets. This results in an immediate drop on the disconnection rate, which then returns to pre-attack levels as the targets reconverge (shown in log-scale in Figure 5.1.b to provide a better view of these dynamics). Figure 5.2.a shows the peers returning to their pre-target degree levels: the SODAP biomass degree reflects the changes in parent targets (most peers return to a target of 2), while the average degree of the network as a whole (considering superpeers) converges to a mean of just over 6.

5.3. Varying Churn Levels. The performance of SODAP was evaluated under a range of network churn levels. Values from 0% (no churn, with the only network dynamism resulting from attacks) to 4% (which is very considerable churn) were tested. Figure 5.3 shows the effects on the disconnection rate. Both protocol versions remain well-connected when there is no churn, as expected (the small spike near the beginning of the experimental runs for basic Myconet is caused by some nodes being dropped by their neighbors due to normal protocol operations, as the network explores in order to find the most efficient topology). Figure 5.4.a shows that in the absence of churn the parent target values for SODAP nodes remain at one until the attack is detected (Myconet biomass nodes always maintain only a single parent).

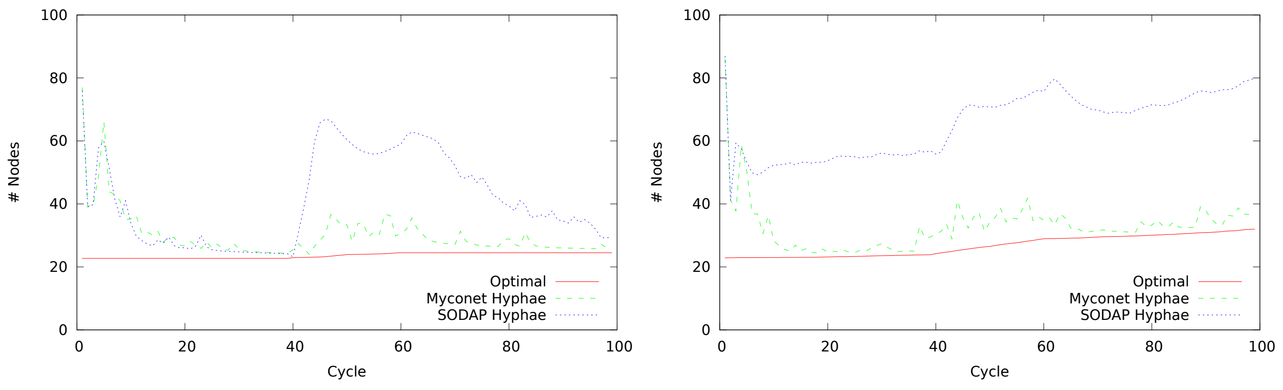
Log-scale Figures 5.3.a and 5.3.b show how the basic Myconet protocol is much more sensitive than SODAP to increasing churn. Myconet's disconnection rate remains roughly equivalent to the churn rate: a four-times increase in the churn rate results in a four-times increase in disconnections. highlights. In contrast, the SODAP disconnection rate stabilizes at around 0.2–0.3% across all evaluated churn levels.

The effect of the attack on the two protocols is quite different. Myconet experiences a 10-11% disconnection



(a) 0% churn (b) 3% churn

FIG. 5.4. *Mean node degrees, varying churn rate (10,000 nodes, 3% churn, 20-round attack beginning at cycle 40)*



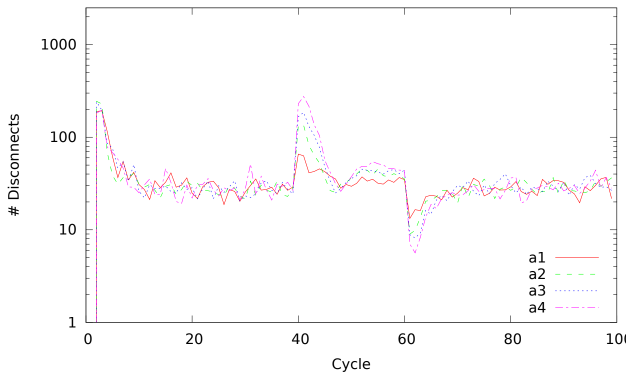
(a) 0% churn (b) 3% churn

FIG. 5.5. *Superpeer counts, varying churn rate (10,000 nodes, 2% churn, 20-round attack beginning at cycle 40)*

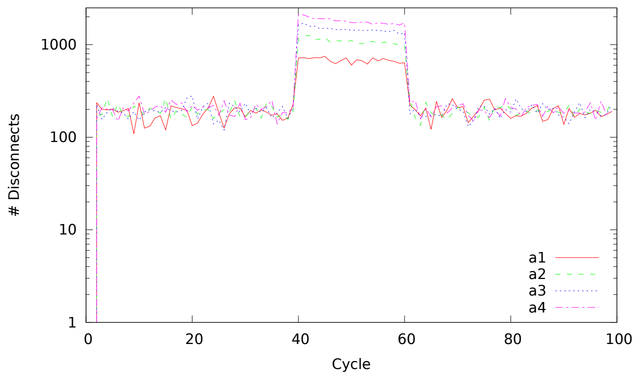
rate as long as the attack continues. The targeted attack also hits the no-churn SODAP network hard at first, with around 10% of the nodes being disconnected for the first two rounds; within a few cycles, though, this drops greatly as SODAP increases the parent targets, converging to around 0.3% (Figure 5.3.a). Networks that have been subjected to higher levels of churn are strengthened against the attack: even with a churn level of 1%, the first two rounds of attack result in a SODAP disconnection rate of less than 2%, and a churn level of 4% results in the attack disconnecting around 1% of nodes for the first two rounds. At all churn levels, SODAP converges to a during-attack disconnection rate of around 0.3–0.4%, slightly higher than the rate during normal operation.

Following the attack, basic Myconet returns to its pre-attack disconnection rate, while SODAP's disconnection rate is actually somewhat improved, particularly at higher churn levels. The reason for this can be seen in Figures 5.4 and 5.5: the biomass target is elevated during the attack (to a mean of just under 3 in both scenarios shown), but then falls off gradually following the attack. The fall-off is much slower for the 0% churn case due to the sigmoid shape of the decay function. A single parent is insufficent to protect from disconnection during churn, but is the ideal state when churn is not present, and the decay rate to a single parent is relatively slower due to the increased risk. The capacity of largest nodes in the network also decreases, on average (shown by the increasing number of peers required to achieve optimality, the solid red line in Figure 5.5). Thus, a proportionally higher number of superpeers must be promoted and maintained.

5.4. Attack Scenarios. We also evaluated the effects of attacks that target a varying number of the highest-degree superpeers (from 1 to 4 per cycle). As Figure 5.6.a shows, the effect of increasing attack size on

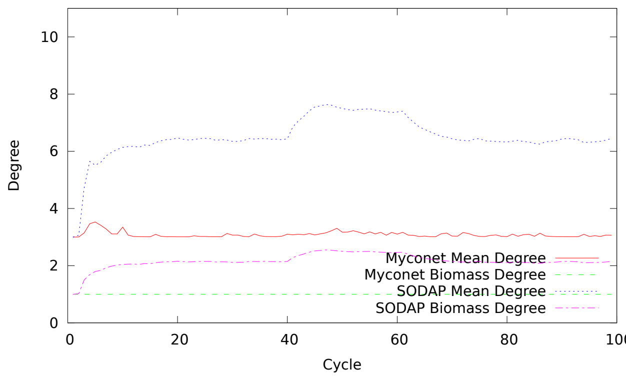


(a) SODAP disconnection rate

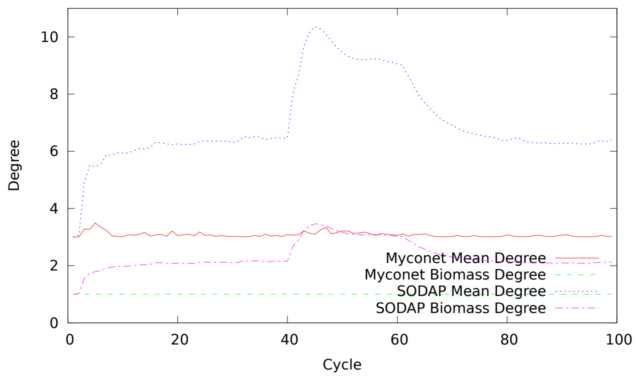


(b) Myconet disconnection rate

FIG. 5.6. *Disconnection rates, varying attack size $1 \leq k \leq 4$ (10,000 peers, 20-round attack)*



(a) Attack $k = 1$



(b) Attack $k = 4$

FIG. 5.7. *Mean node degrees, varying attack size k (10,000 nodes, 2% churn, 20-round attack beginning at cycle 40)*

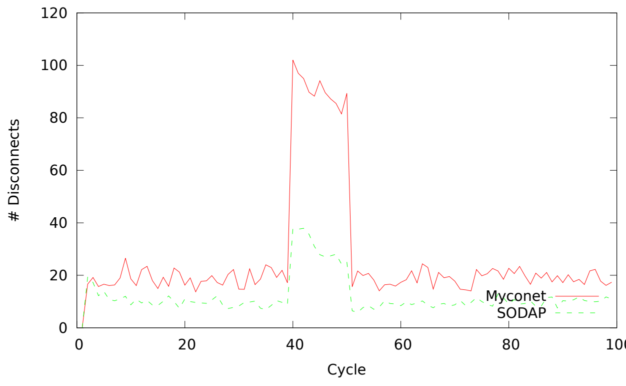
Myconet with SODAP is most visible at the beginning of the attack, determining the size of the initial spike (around 0.5% for an attack size of $k = 1$, up to around 2% for $k = 4$). As the attack continues and the network degree adapts to avoid disconnection, this drops to levels slightly above the disconnection rate for background churn (from around 0.25% for $k = 1$ to around 0.35% for $k = 4$). The behavior of basic Myconet is much less resilient; an attack of $k = 1$ already results in a disconnection rate of around 5%, up to nearly 11% for for $k = 4$.

Following the attack, basic Myconet quickly returns to pre-attack disconnection rates, while SODAP shows much larger dips for larger attacks before stabilizing. This reflects the additional network connections that were established in order to maintain connectivity.

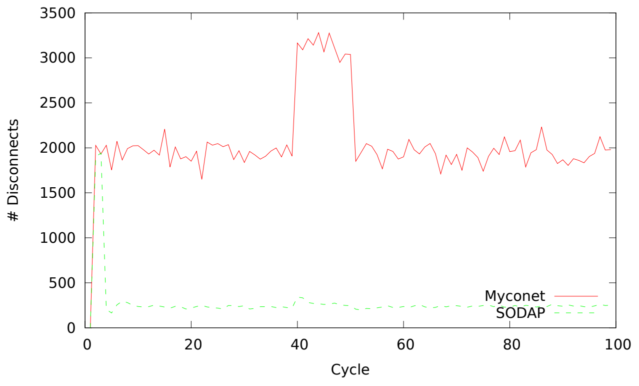
The greater increase in degree by SODAP for larger attacks is shown in Figure 5.7. In response to an attack of $k = 1$, the average network degree increases to around 7.5, returning to pre-event levels following the end of the attack. An attack of $k = 4$, on the other hand, is extremely significant, as many nodes are likely to be affected by the abrupt removal of this many high-degree superpeers. To protect against this, SODAP initially raises the mean node degree to around 10, stabilizing at around 9 as the attack continues. After the attack ends, the average degree drops as the parent targets decay.

5.5. Network Scale. The performance of SODAP at different network scales was also evaluated, from 10^3 to 10^5. These results are shown in Figures 5.8 and 5.9 (results for 10^4 are found in Section 5.2, above).

For networks of 1,000 nodes (with $c_{max} = 50$) and 2% churn, the basic Myconet protocol stabilizes at a disconnection rate of around 2%. SODAP reduces this to about 1%. SODAP also reduces the increase in disconnection rate during an attack, from around 9% for basic Myconet to around 3.5%, which is comparable

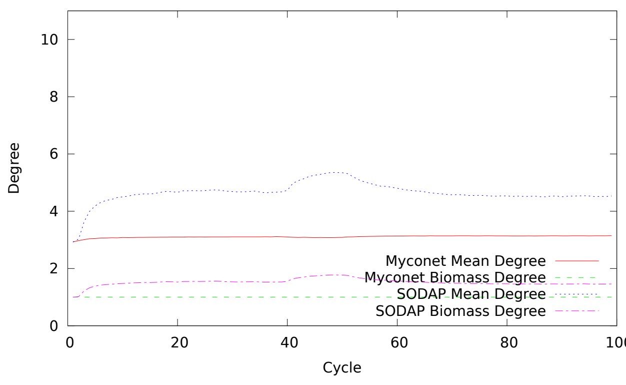


(a) 1,000 nodes

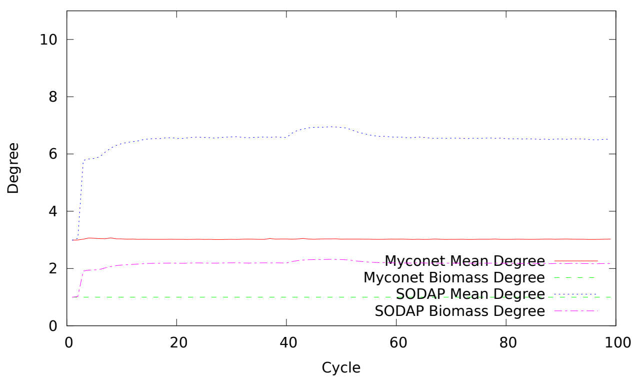


(b) 100,000 nodes

FIG. 5.8. *Disconnection rates, varying network size (2% churn, 10-round attack)*



(a) 1,000 nodes



(b) 100,000 nodes

FIG. 5.9. *Mean node degrees, varying network size k (2% churn, 10-round attack)*

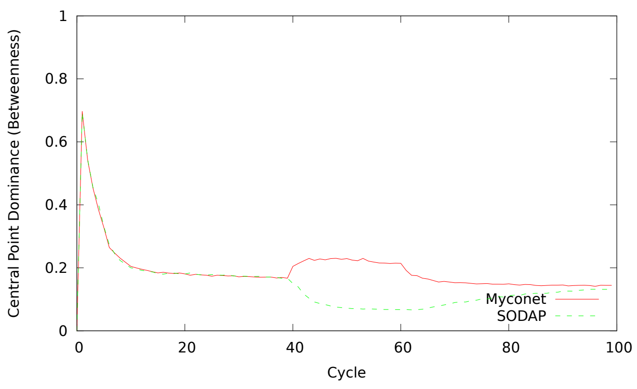
to the 10,000 node case.

At 100,000 nodes, Myconet's disconnection rate remains proportional at 2%, while Myconet with SODAP begins at this level but quickly drops to around 0.25%. Targeted attacks on the two largest superpeers ($k = 2$) have, of course, a much lesser effect at this larger scale. During such an attack, basic Myconet jumps to a rate of around 3% (versus 11% for 10^4). SODAP's response is comparable to the smaller network, rising slightly to around 0.3%, as compared to 0.25% for 10^4.

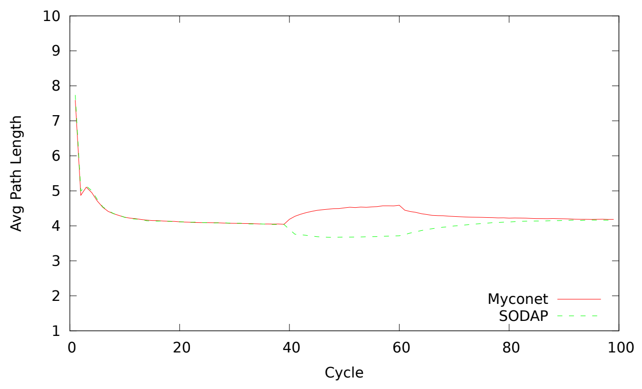
The effect that SODAP has on node degrees is shown in Figure 5.9. For networks of 10^3 nodes, a mean parent target of around 1.5 is selected, which rises to close to 2 during the attack. This is reflected in the overall difference in mean degree for all nodes in the network, which is around 4.7 (compared to 6.2 for 10^4). At 10^5, the results are slightly higher than those for 10^4, with a mean parent target of just above 2 and mean degree around 6.5.

5.6. Other Metrics. Figures 5.10 and 5.11 show the central point dominance (CPD) of the largest connected component (using betweenness centrality) [7] and average path lengths for experimental runs with 10^3 nodes.

As can be see in Figure 5.10.a, without churn, these metrics remain relatively close until the targeted attack begins at round 40. At this point, the degree of the leaf peers rises when SODAP is used, resulting in a drop in the level of centralization in the network. Similarly, Figure 5.10.b shows that this results in slight drop in average path length, as peers' parent targets increase. Without SODAP, the average path length increases during the attack.

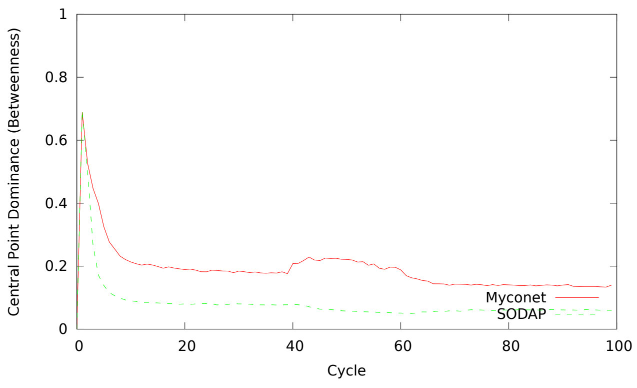


(a) Central Point Dominance (Betweenness)

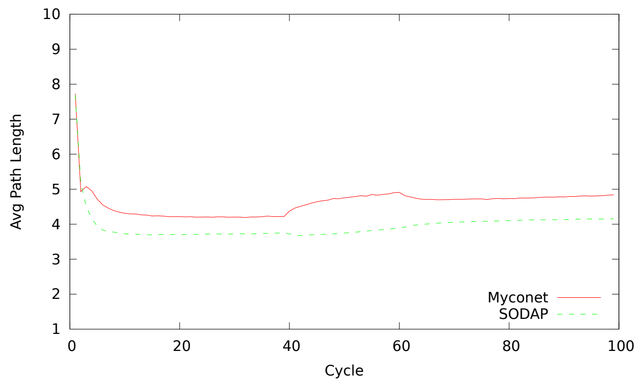


(b) Average Path Length

FIG. 5.10. *Graph metrics for network with 1,000 nodes, no churn, 20-round attack beginning at cycle 40*



(a) Central Point Dominance (Betweenness)



(b) Average Path Length

FIG. 5.11. *Graph metrics for network with 1,000 nodes, 2% churn, 20-round attack beginning at cycle 40*

With 2% churn, the difference in central point dominance is readily apparent in Figure 5.11.a; with SODAP, it is lower due to the higher average degree of the leaf peers. CPD drops slightly further during the attack, and slowly returns to pre-attack levels after its conclusion as the parent targets decay. Figure 5.11.b shows average path lengths; with SODAP, path lengths remain around 4 before, during and after the attack (though it can be seen that the level following the attack is slightly higher due to the decreased maximum capacity from targeted removal of those peers). Without SODAP's degree adaptation, the path length rises during the attack, and settles to a slightly higher level after it ends.

6. Conclusions. In this paper, we have proposed the *SODAP* strategy for the self-organized protection of unstructured superpeer overlays from topology attacks. SODAP effectively protects the overlay from targeted attacks against high-degree nodes, which can be a significant vulnerability of superpeer-based P2P networks.

We evaluated our approach using a modified version of the Myconet superpeer overlay, and compared its performance with the non-adaptive approach. The experimental results show that, relative to the non-degree-adaptive approach, SODAP provides a very significant reduction (around 90%) in peer disconnection rate for networks under attack, across a range of network sizes and conditions.

A key advantage of SODAP is its ability to seamlessly adapt in both directions, increasing node degrees in response to detected disconnections, but also decreasing them in the absence of failures. SODAP improves previous work in this area, and is effective across multiple network scales, churn levels, and attack sizes, as well as adapting smoothly (through the exploratory reduction in degree of a small number of peers) to improvements in network conditions without requiring the reversion of the entire topology.

SODAP's self-protection mechanism has a smaller parameter space than previous work, making it more flexible and easier to apply and tune. An in-depth examination of the the effects of varying SODAP's rules and parameters can be found in [24].

The protocol focuses on disconnections (rather than attempting to guess whether an attack is underway), an approach that has the advantage of being directly, locally observable. Since SODAP nodes attempt to minimize disconnections no matter what their cause, the problem of possible false positives and false negatives does not need to be considered.

Though the implementation in this paper is built on top of Myconet, SODAP is applicable to any unstructured peer-to-peer network that uses a superpeer topology, independent of the protocol that is used for superpeer selection. One key property of SODAP is that it exploits heterogeneity that typically occurs among overlay participants, in particular at large scale, and is designed to enable the superpeer network to continue taking advantage each peer's individual capabilities, even when under targeted attack. The selected, high-degree superpeers are leveraged in SODAP itself, as they act as both connection points for reconnecting peers and conduits for announcing these reconnections to a relatively large number of other peers; in this way, a few disconnection messages are able to spread through a system that is both attempting to minimize disconnections and minimize the degree of leaf peers, which helps improve efficiency.

6.1. Future Work. We plan to compare this adaptive mechanism against topology preservation overlays that use a fixed number for the parent target value of leaf peers. We also plan to test the performance of SODAP outside of a simulation environment and in a live network environment, using our recent Protopeer-based implementation of Myconet.

Balancing the upward and downward pressure on node degrees results in a disconnection rate that is very low, but is not zero. Some inefficiency (in the form of actual disconnected nodes) is inevitable, and in fact needed to keep the SODAP system stabilized at a particular level of parent targets. A possible improvement may be to have nodes adjust as a result of "near" disconnections (*i.e.*, if they were reduced to a single neighbor within the time window). This may allow a larger percentage of nodes to avoid disconnection at the price of slightly higher average degree.

The new topologies constructed by the local actions of peers (increasing the degree of leaf peers by adding additional superpeer connections) are influenced by the underlying overlay protocol. One direction for future experiments is to apply SODAP's strategy to a superpeer-based overlay other than Myconet. Another is to examine additional neighbor-selection strategies for constructing the reinforced topologies.

SODAP focuses on self-protection from attacks that result in disconnection of peers. Many other types of attacks against peer-to-peer networks have been examined in the literature, such as those discussed in the attack taxonomy in [31]. Direct attacks against learning mechanisms or the self-organized behavior itself are also possible: for example, peers might report false values for disconnections, or for their current parent targets. We would like to examine the effect of these different types of attacks on our strategy. Moreover, within the bounds of the discussed attack itself, we also plan to perform more detailed analysis of the trade-offs between sensitivity to attacks and the inefficiency induced by the increase in node degrees.

REFERENCES

[1] S. Baset and H. Schulzrinne, *An analysis of the skype peer-to-peer internet telephony protocol*, in INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings, April 2006, pp. 1–11.

[2] N. Basher, A. Mahanti, A. Mahanti, C. Williamson, and M. Arlitt, *A comparative analysis of web and peer-to-peer traffic*, in Proceedings of the 17th international conference on World Wide Web, ACM, 2008, pp. 287–296.

[3] B. Beverly Yang and H. Garcia-Molina, *Designing a super-peer network*, in Data Engineering, 2003. Proceedings. 19th International Conference on, 2003, pp. 49–60.

[4] M. Brinkmeier, G. Schafer, and T. Strufe, *Optimally DoS resistant P2P topologies for live multimedia streaming*, Parallel and Distributed Systems, IEEE Transactions on, 20 (2009), pp. 831–844.

[5] C. Diwakar, *Security threats in peer to peer networks*, Journal of Global Research in Computer Science, 2 (2011).

[6] M. Engle and J. I. Khan, *Vulnerabilities of P2P systems and a critical look at their solutions*, Tech. Report TR2006-11-01, Kent State University Medianet Lab, 2006.

[7] L. Freeman, *A set of measures of centrality based on betweenness*, Sociometry, 40 (1977), pp. 35–41.

[8] W. Galuba, K. Aberer, Z. Despotovic, and W. Kellerer, *Protopeer: a p2p toolkit bridging the gap between simulation and live deployement*, in Proceedings of the 2nd International Conference on Simulation Tools and Techniques, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, p. 60.

[9] C. G. Ghedini and C. H. Ribeiro, *Rethinking failure and attack tolerance assessment in complex networks*, Physica A: Statistical Mechanics and its Applications, 390 (2011), pp. 4684–4691.

[10] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris, *The Peersim simulator.* http://peersim.sf.net.

[11] P. Keyani, B. Larson, and M. Senthil, *Peer pressure: Distributed recovery from attacks in peer-to-peer systems*, in Revised Papers from the NETWORKING 2002 Workshops on Web Engineering and Peer-to-Peer Computing, London, UK, UK, 2002, Springer-Verlag, pp. 306–320.

[12] P. Lewis, H. Goldingay, and V. Nallur, *It's good to be different: Diversity, heterogeneity, and dynamics in collective systems*, in Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2014 IEEE Eighth International Conference on, Sept 2014, pp. 84–89.

[13] W. Liu, J. Yu, J. Song, X. Lan, and B. Cao, *Erasp: An efficient and robust adaptive superpeer overlay network*, in Progress in WWW Research and Development, Y. Zhang, G. Yu, E. Bertino, and G. Xu, eds., vol. 4976 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 468–474.

[14] D. Lucia, *Mycocloud: Improving QoS by managing elasticity of services in decentralized clouds*, master's thesis, 2013.

[15] B. Mitra, M. Afaque, S. Ghose, and N. Ganguly, *Developing analytical framework to measure robustness of peer-to-peer networks*, in Distributed Computing and Networking, S. Chaudhuri, S. Das, H. Paul, and S. Tirthapura, eds., vol. 4308 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006, pp. 257–268.

[16] B. Mitra, A. Dubey, S. Ghose, and N. Ganguly, *Formal understanding of the emergence of superpeer networks: A complex network approach*, in Distributed Computing and Networking, K. Kant, S. Pemmaraju, K. Sivalingam, and J. Wu, eds., vol. 5935 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2010, pp. 219–230.

[17] B. Mitra, F. Peruani, S. Ghose, and N. Ganguly, *Analyzing the vulnerability of superpeer networks against attack*, in Proceedings of the 14th ACM conference on Computer and communications security, ACM, 2007, pp. 225–234.

[18] B. Mitra, F. Peruani, S. Ghose, and N. Ganguly, *Measuring robustness of superpeer topologies*, in Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing, PODC '07, New York, NY, USA, 2007, ACM, pp. 372–373.

[19] A. Montresor, *A robust protocol for building superpeer overlay topologies*, in Proceedings of the 4th IEEE International Conference on Peer-to-Peer Computing, Zurich, Switzerland, Aug. 2004, IEEE, pp. 202–209.

[20] P. Perlegos, *DoS defense in structured peer-to-peer networks*, Tech. Report UCB/CSD-04-1309, University of California, Berkeley, 2004.

[21] N. Ramzan, H. Park, and E. Izquierdo, *Video streaming over p2p networks: Challenges and opportunities*, Image Communication, 27 (2012), pp. 401–411.

[22] K. Samant and S. Bhattacharyya, *Topology, search, and fault tolerance in unstructured P2P networks*, in System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on, 2004, pp. 6 pp.–.

[23] G. Shi, Y. Long, H. Gong, C. Wan, C. Yu, X. Yang, and H. Zhang, *A high scalability p2p simulation framework with measured realistic network layer support*, in Performance, Computing and Communications Conference, 2008. IPCCC 2008. IEEE International, IEEE, 2008, pp. 311–318.

[24] P. L. Snyder, *Modeling and Engineering Self-Organization in Complex Software Systems*, PhD thesis, Drexel University, 12 2013.

[25] P. L. Snyder, R. Greenstadt, and G. Valetto, *Myconet: A fungi-inspired model for superpeer-based peer-to-peer overlay topologies*, in Self-Adaptive and Self-Organizing Systems, 2009. SASO '09. Third IEEE International Conference on, 2009, pp. 40–50.

[26] P. L. Snyder, Y. Osmanlioglu, and G. Valetto, *Biologically inspired attack detection in superpeer-based P2P overlay networks*, Bio-Inspired Models of Networks, Information, and Computing Systems, (2012), pp. 99–114.

[27] A. Srivastava, B. Mitra, F. Peruani, and N. Ganguly, *Attacks on correlated peer-to-peer networks: An analytical study*, in Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on, 2011, pp. 1076–1081.

[28] K. Suto, H. Nishiyama, N. Kato, T. Nakachi, T. Fujii, and A. Takahara, *Thup: A p2p network robust to churn and dos attack based on bimodal degree distribution*, IEEE Journal on Selected Areas in Communications, 31 (2013), pp. 247–256.

[29] G. Valetto, P. L. Snyder, D. J. Dubois, E. Di Nitto, and N. M. Calcavecchia, *A self-organized load-balancing algorithm for overlay-based decentralized service networks*, in Self-Adaptive and Self-Organizing Systems (SASO), 2011 Fifth IEEE International Conference on, 2011, pp. 168–177.

[30] T. Yi-Hong, L. Xi-Dao, L. Ya-Ping, and Z. Bi-hai, *DLPSPN: New efficient super-peer network based on double-loop Petersen graph*, in Network Computing and Information Security (NCIS), 2011 International Conference on, vol. 2, IEEE, 2011, pp. 201–205.

[31] X. Yue, X. Qiu, Y. Ji, and C. Zhang, *P2P attack taxonomy and relationship analysis*, in Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on, vol. 02, feb. 2009, pp. 1207–1210.

[32] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz, *Tapestry: A resilient global-scale overlay for service deployment*, Selected Areas in Communications, IEEE Journal on, 22 (2004), pp. 41–53.

[33] K. A. Zweig and K. Zimmermann, *Wanderer between the worlds - self-organized network stability in attack and random failure scenarios*, in Self-Adaptive and Self-Organizing Systems, 2008. SASO'08. Second IEEE International Conference on, IEEE, 2008, pp. 309–318.

Edited by: Giacomo Cabri and Emma Hart
Received: Dec 23, 2014
Accepted: Jul 15, 2015

DOI 10.12694/scpe.v16i3.1102
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 16, Number 3, pp. 289–301. http://www.scpe.org © 2015 SCPE

CLASSIFYING AND FILTERING USERS BY SIMILARITY MEASURES FOR TRUST MANAGEMENT IN CLOUD ENVIRONMENT

FATIMA ZOHRA FILALI[*] AND BELABBAS YAGOUBI[†]

Abstract. Trust represents an important issue for adopting cloud services. A trust management framework is essentially, about user rating. Hence, correctly addressing user feedback and filtering out malicious rating is a main step in providing reliable services. In order to process their feedback and calculate a reliable trust degree. Thus, new opportunities can be offered for the establishment of a trust relationship among involved entities.

We propose a technique to process user rating by statistical methods. Then, we proceed to classify the users into different groups to detect malicious users. The users are grouped according to their rating by a k-means clustering technique, and the evaluation will show that the proposed solution gives better results than the traditional filtering solution.

Key words: Cloud, Trust Management, Similarity Measure, Filtering, Rating, Distance, Threats, Malicious, Fair, Unfair

AMS subject classifications. 68M14

1. Introduction.

For the last years, Cloud services have grown to become an essential paradigm for both industry and academia, by allowing Cloud consumers to rent Computing, network, and storage resources. In that way, the consumers pay for their use of services without apprehensions about maintenance, management or cost.

In spite of all importance of Cloud Computing, most of the organizations are not making a trend of it, and its evolution has raised many concerns and was encountered by various obstacles. Security is one of the most crucial problems for this model, and the risks accompanying the deployment of services and applications are more important with the architecture of Cloud environment [1]. Moreover, in cloud computing environments, the user is dependent on the provider for various services. For many services, the user has to trust the provider for storing his data. Thus, a trust framework should be developed to allow trust establishment, interaction management or requirements share.

With the development of Cloud computing, we are heading towards more distributed and highly available infrastructures, which contribute significantly to the deployment of services anywhere, at anytime and for anyone. To provide reliable services in such infrastructures, we should consider both user's feedback and trust requirements within Cloud environment. Designing a trust management framework requires one to understand what service an entity needs, how to provide reliable feedback from different users, which services can be trusted, how categorize different users to assess feedback, etc.

In a previous work [2], we have proposed a trust model for Cloud computing, based on an opinion model for the subjective dimension and performance parameters for the objective dimension. The proposed model was validated by simulation with well-known trust model. Then, the trust model was integrated in a framework for trust management in Cloud Computing environment [3]. The designed framework identifies the metrics of performance to select the most suitable provider for performing service transaction and integrate it into trust rating process while filtering biased opinions with a simple bias function. This paper is a continuation of these works by improving the credibility evaluator to filter malicious raters.

Hence, this research describes the threats associated with trust management, especially for Cloud environment. Then, we categorize the users according to these threats into four classes. After that, we discuss the issue of filtering biased users, and the different statistical measures and distance. Next, we propose a method for computing the similarities between users. We also describe a mechanism to integrate a reward/punish service to the trust management framework. The next section, show experiments and results of the proposed solution. Finally, we discuss the future work.

2. Related Work.

Cloud Security Alliance [4] have identified various security threats to cloud computing. In [5], the author classify several vulnerabilities and attacks that can be encountered in Cloud Computing, at

[*]Department of Computer Science, University of Oran1 Ahmed Ben Bella, Oran, Algeria (filalifz7@gmail.com).
[†]Department of Computer Science, University of Oran1 Ahmed Ben Bella, Oran, Algeria (byagoubi@gmail.com).

different security points such as : Abuse and Nefarious Use of Cloud, Insecure Interfaces and APIs, Malicious Insider, Virtualized Technology... However, these threats represent various aspects of security. To ensure the proper implementation of a trust management framework, specific threats must be considered. Hence, we focus on behaviour attacks in Cloud Computing.

For a trust management framework, a vulnerability is a weakness in the system that could be exploited to influence the recommendation and the trust of the provided services. Cloud computing is just as vulnerable as any other technology that uses the public internet for connectivity. The vulnerabilities include malicious attack, a man-in-the-middle attack, Sybil attack, denial-of-service attack, etc. Furthermore, Various types of rating attack against the trust management systems such as Bad mouthing attack, Feedback Collusion, Sybil attack, Reputation lag attack have been discussed in [6, 7, 8, 9, 10, 11].

In many researches, it has been demonstrated that users with false feedback have some common features. In [10], it has been identified that malicious users who have similar characteristics such as a higher request frequency to surpass honest users. As, users who try to increase or decrease a service popularity submit feedbacks frequently. They also tend to usually engage in minimum value transactions to meet the requirements of submitting a rating. Furthermore, malicious ratings tend to be either significantly lower or higher than the majority of the users [13]. Thus, a trust management system should have the ability to weigh the ratings of highly credible users more than those with a low credibility rating. There are several approaches that evaluate trustworthiness of users based on majority opinion, such as beta filtering feedback [14]. This approach works as long as the majority of ratings are not from a group of users that tend to falsify their ratings.

Another approach that uses beta probability density function to estimate the reputation of a provider as either bad or good is discussed in [15].This approach was later extended such that a feedback is considered to be fair if it falls in the range of lower and upper boundaries among all the ratings [9]. The limitation of this strategy is that users could collude as a group to manipulate the majority ratings. However, majority ratings scheme alone is not sufficient to accurately measure the trustworthiness of a user.

The authors [16] proposed models based on assumption that all customers in the system have provided feedbacks for a given period of time. For example, new users could be treated as bad users and their feedback will carry less weight in trust assessment.

Most of these approaches are based on similarity filtering technique such as [9, 14]. In this technique, the users with low similarity rating are considered as less truthful. These approaches have been proved very effective in term of filtering malicious users. Hence, we are going to use a similarity-based technique to proceed to a first phase which will consist at predicting missing values for rating services. We point out that the focus of our study is not computing the degree of trust but filter out malicious users to provide the trust management framework with reliable rating.

3. Cloud Computing trust threats. In this section, we will discuss some threats and vulnerabilities about trust management system. Various researchers [9, 11, 17] have discussed rating attacks and threats against trust management systems. We will present and describe the most common security threats applicable in the field of trust and reputation management over Cloud computing environments:

Cold-start problem: reference the issue where new services or new user recommendations meet difficulties to provide an adequate rating for the system. A service cannot be recommended unless it has a sufficient number of user ratings. As other users in the system tend to interact with high reputable services, the chance of a new service being selected for interaction is generally rare [18].

Malicious feedback: reference the issue where users report falsely their feedback, creating errors in the system. These malicious feedbacks can be either individual malicious where a unique user always provide false feedback for services or collective malicious when two of more users collectively boost a service reputation or conspire against a service provider. Hence, a trust management system must exclude unfair ratings. A common solution consists to use their statistical properties [19, 20].

Playbooks: a playbook consists to maximize the profit of a service according to certain criteria. For example, a provider can act honestly and provide quality services over a period to gain a high reputation score, then with his high reputation score providing low-quality services at a low production cost [21]. Thus, a trust management system must consider the property of auto-adjusting of the service rating over time.

Sybil attack: malicious users may acquire multiple identities. Each time it provides a false feedback in the system, he replaced it with a new identity. For that propose, a good trust framework must integrate a reliable authentication service based not only on security but also on filtering out phantom identity. For our present work, we didn't consider this kind of attacks. From the discussed threats and vulnerabilities, we can observe that generally four groups of users can be categorized, either for a user or for a provider of the service. Hence, we assume four main group in our work:

1. Fair Positive (FP), when an entity is providing honest rating about a service with a high quality.
2. Fair Negative (FN), when an entity is providing honest rating about a service with a low quality.
3. Unfair Positive (UP), when an entity is providing malicious rating about a service with a high quality (providing unreasonably increased feedback assessments).
4. Unfair Negative, when an entity is providing malicious rating about a service with a low quality (providing low feedback assessments).

We also observe that the principal part of providing a reliable trust management system is to correctly identify the unfair rating. To achieve that it is essential to correctly identify in which groups the user is classified.

Hence, in this paper we propose to group users into four groups by using a learning technique such as k-means. The problem with that method is the missing value for many users. Consequently, before classifying groups we treat the problem of cold start by using a statistical method. Finally, as it has been pointed out in [22] we integrate a service for rewarding honest users groups and punish malicious groups.

4. Measuring rating similarities.

The users of different services may provide their feedbacks about the services they consumed to present their satisfaction or dissatisfaction. Based on the provided feedbacks, the system re-evaluates trust rates for the services and service providers. However, the users may provide unfair and false feedback, so it is important to detect such ratings. Feedback filtering component is necessary to address the discussed threats. Some techniques to prevent unfair feedback have been presented in different studies [11, 12, 13, 14]. Although there are many algorithms to filter the unfair rating, the basic idea is to calculate different measures of similarity between users.

There are several similarities algorithms [23, 24] that have been used in filtering field Pearson correlation, cosine vector similarity, Euclidean distance and Minkowski distance. These measures can be effectively used to balance the prediction algorithm in the meaning of the ratings, therefore, to improve accuracy.

4.1. Pearson's correlation.

Pearson's correlation measures the linear correlation between two sequences of ratings for the users x and the user y about the set of services m rated by both user x and user y.

$$Sim\_Pearson(x,y) = \frac{\sum_{i=1}^{m}(R_{ix}-\overline{R_{ix}})(R_{iy}-\overline{R_{iy}})}{\sqrt{\sum_{i=1}^{m}(R_{ix}-\overline{R_{ix}})^2\sum_{i=1}^{m}(R_{iy}-\overline{R_{iy}})^2}} \tag{4.1}$$

where R_{ix} is the rating of the service i by the user x, $\overline{R_{ix}}$ is the average rating of user x, and R_{iy} is the rating of the service i by the user y, $\overline{R_{iy}}$ is the average rating of user y.

4.2. Cosine measure.

The cosine measure looks at the angle between two sequences of ratings where a greater similarity imply smaller angle, as following formula:

$$Sim\_Cosine(x,y) = \frac{\sum_{i=1}^{m} R_{ix}R_{iy}}{\sqrt{\sum_{i=1}^{m} R_{ix}^2 \sum_{i=1}^{m} R_{iy}2}} \tag{4.2}$$

where R_{ix} is the rating of the service i by the user x, $\overline{R_{ix}}$ is the average rating of user x, and R_{iy} is the rating of the service i by the user y, $\overline{R_{iy}}$ is the average rating of user y.

4.3. Euclidean distance.

A Euclidean distance represents the distance between two points in Euclidean space. We are going to use this distance to measure the distance between sequences of ratings for two users x and y about m services rated by both users.

$$Dis\_Euclidean(x,y) = \sqrt{\sum_{i=1}^{m}(R_{ix}-R_{iy})^2} \tag{4.3}$$

where R_{ix} is the rating of the service i by the user x, and R_{iy} is the rating of the service i by the user y.

4.4. Minkowski distance. The Minkowski distance can be considered as a generalization of both the Euclidean distance and the Manhattan distance. We used this metric to measure the distance between two sequences of ratings.

$$Dis\_Minkowski(x,y) = (\sum_{i=1}^{m}(R_{ix} - R_{iy})^p)^{\frac{1}{p}} \tag{4.4}$$

where R_{ix} is the rating of the service i by the user x, and R_{iy} is the rating of the service i by the user y.

4.5. Hamming Distance. The Distance of Hamming is used in information theory, to measure the difference between two set of strings or two sequence of bits. In another way, it measures the minimum number of errors that could have transformed one string into the other.

We are going to use this distance to measure the difference in two sequence of ratings, as described in the following formulas:

$$Dis\_Hamming(x,y) = \sum_{i=1}^{m}[R_{ix} \neq R_{iy}] \tag{4.5}$$

In this equation $Dis\_Hamming(x,y)$ is the Hamming distance between the user x and the user y, i is the index of the service rated for a total of n services. The Hamming distance gives the number of mismatches between the rating of user x and user y.

5. The proposed solution. For assess a reliable trust degree, it is important to filter unfair rating for the different services. Literature demonstrates that filtering methods successfully provide abundant evidence. However, there are some ways inadequate since the various filtering methods has cold start problems.

In this paper, we propose to classify users in groups. For this propose, we use the k-means clustering algorithm [25], to form user groups depending on the rating. However, the machine learning techniques suppose the presence of full knowledge about the user rating for different services. However, in a cloud environment and especially for a trust management system, these suppositions do not provide accurate results. Many of statistical literature deals with this case by replacing randomly the missing data, which will result in an inaccurate and biased estimator.

For that reason, we propose to pre-process the set of rating before proceeding to the classification, and since statistical techniques offer better results, we propose a hybrid technique based on the different measures discussed in the previous section.

We employ the neighbours to help identify users' classes. We find that the neighbour group mean deviation is small when the current service is objective, while it is large when the current service is subjective. Then, the resulting classes are subject to either recompensing for the fair groups or penalizing for the unfair groups.

5.1. Algorithm. We consider a set of users with fairly positive feedbacks, fairly negative feedbacks, unfairly positive feedbacks and unfairly negative feedbacks. For each user i, we have a sequence of Ni ratings about different services. The classification algorithm involves that the user rate all used services. Each user is represented by the term rating R_{ij} in the user list sequence, which contains the ratings have been expressed by a user i for the service j, as described in the following table.

Like discussed before, in a real environment it is impossible for a user to use each proposed service. Therefore, the presented table will contain many missing values for different service, which will result in an accurate or biased classification.

For that reason, we proceed to a pre-process phase where we predict the missing values by using the discussed statistical measures in the previous section. The resulting matrix values are then classified into four clusters using the k-means clustering algorithm [25]. We employ the resulting neighbors to help identify users' classes. We assume that the rating set whose mean is closer to the mean of neighbors' ratings may be from honest peer. Among these four sets, the rating set with the least group mean deviation may be the dishonest ones.

Table 5.1
Matrix of Users Rating

Users & Services	S_1	S_2	...	S_m
U_1	R_{11}	R_{21}	...	R_{m1}
U_2	R_{21}	R_{22}	...	R_{m2}
...	...	...	...	...
U_n	R_{n1}	R_{n2}	...	Rnm

Note that this solution is proposed in the context of trust-dependent assumption that a service rating for a user is dependent on provided quality of service. The procedures of the proposed solution are as follows.

Step 1: Select the neighbours for each user

Algorithm 1 select_neighbors (user x)

```
Begin
    for  $user\ y \neq x$  do
        $d \leftarrow Dis\_Hamming(x, y)$                                  ▷ Formula 4.5
        if $d < 0.5$ then
            $add\ y\ to\ neighbors\_list$
        end if
    end for
    return neighbors_list
End
```

Step 2: Calculate the similarities between users

Algorithm 2 get_similar_users (user x)

```
Begin
    for  $user\ y \neq x$  do
        $s_1 \leftarrow Sim\_Pearson(x, y)$                                ▷ Formula 4.1
        $s_2 \leftarrow Sim\_Cosine(x, y)$                                 ▷ Formula 4.2
        $d_1 \leftarrow Dis\_Euclidean(x, y)$                             ▷ Formula 4.3
        $d_2 \leftarrow Dis\_Minkowski(x, y)$                             ▷ Formula 4.4
        if $(s_1 \geq 0.5) AND (s_2 \geq 0.5) AND (d_1 \leq 0.5) AND (d_2 \leq 0.5)$ then
            $add\ y\ to\ similar\_users\_list$
        end if
    end for
    return similar_users_list
End
```

Step 3: Compute the predicted missing rates for each user

Step 4: Classify users The resulting matrix values of rating are then classified into four clusters using the k-means clustering algorithm: FP, FN, UP, UN.

5.2. Self-Adjusting classification. Self-adjusting is an important issue where the user groups are established dynamically to reflect recent interactions and rating. The proposed solution would detect unfair users, changes in rating rates, and reforms new groups of users, by periodically performing clustering and detecting new groups of users.

5.3. Punishment and Reward. According to the resulted groups, a last step of punishment or recompense has to be performed. This punishing or rewarding mechanism works by decreasing or increasing, respectively, the weights assigned to each source of rating, which will depend on the user groups.

Table 6.2
Rating models

		Reference Classes						
Filtering techniques	Users classes	FP	FN	UP	UN	Total Predicted	Accuracy % Acc.	MAE
Predicted classes	FP	**136**	2	10	1	149	91%	0.180
	FN	13	**146**	0	5	164	97%	0.147
	UP	1	0	**86**	10	97	86%	0.250
	UN	0	2	4	**84**	90	84%	0.220
	Total	150	150	100	100	500	90%	0.200
Overall Accuracy	452/500 = 90%							

with r_1, r_2, r_3, ..., r_n is the prediction of users' ratings, and c_1, c_2, c_3, ..., c_n is the corresponding real rating data set of users.

The metric RMSE is defined as:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(r_i - c_i)^2}{n}} \tag{6.2}$$

The lower the MAE and RMSE, the more accurate the predictions would be.

In the same way, we used this two metrics to evaluate the accuracy of the classified groups.

To evaluate the prediction accuracy of our approach, we compare it with the two following approaches:

- Random: before proceeding to groups clustering, the missing values are generated randomly.
- Mean: the missing value for a user x about a service i is substituted by the value of rating mean for the user x.

6.3. Experiments. The simulation environment consists of cloud service users and cloud service providers. The simulation proceeds in simulation cycles. Each simulation cycle cloud users proceed to service rating corresponding to the model detailed in Table 6.2.
The results reported in this section were obtained assuming that 30% of the users are positive fair, 30% are positive unfair, 20% are positively biased, and 20% are negatively biased. In addition, we assumed that the number of ratings NR =10, with 1 being the lowest and 10 being the highest. (The selection of NR=10 is not significant, and any other values can be readily used).

6.3.1. Experiment 1. In this set of experiments, the total number of users is 500, and the total number of services is 25.

The results indicate that the proposed solution obtained high values for classification accuracy (84% - 97%) for each class of users (Table 6.2). The table shows that the overall classification accuracy of the proposed approach was 90%, which represents a high value.

For more model validation, we have conducted a repeated random sampling for 10 times for the proposed solution with the random filtering and the mean filtering. The experimental results of the accuracy is shown in Figure 6.1 for Fair Positive, Figure 6.2 for Fair Negative, Figure 6.3 for Unfair Positive, and Figure 6.4 for Unfair Negative. These figures shows that the proposed solution obtained the best classification accuracy representing for all the simulation sounds.

We also present the best, average and worst cases for each techniques, in Table 6.3.

From the results of the three groups of experiments, we can see that in the best case, our solution improves over Random Filtering and Mean filtering techniques respectively by 56%-15% in term of MAE and by 47%-29% in term of RMSE. In the worst case, the improvements increase to 61%-45% in term of MAE and 49%-29% in term of RMSE. This result means that our model has better robustness. In other words, it not only performs well in the best case but also overcomes the worst-case situations with slightly lower accuracy.

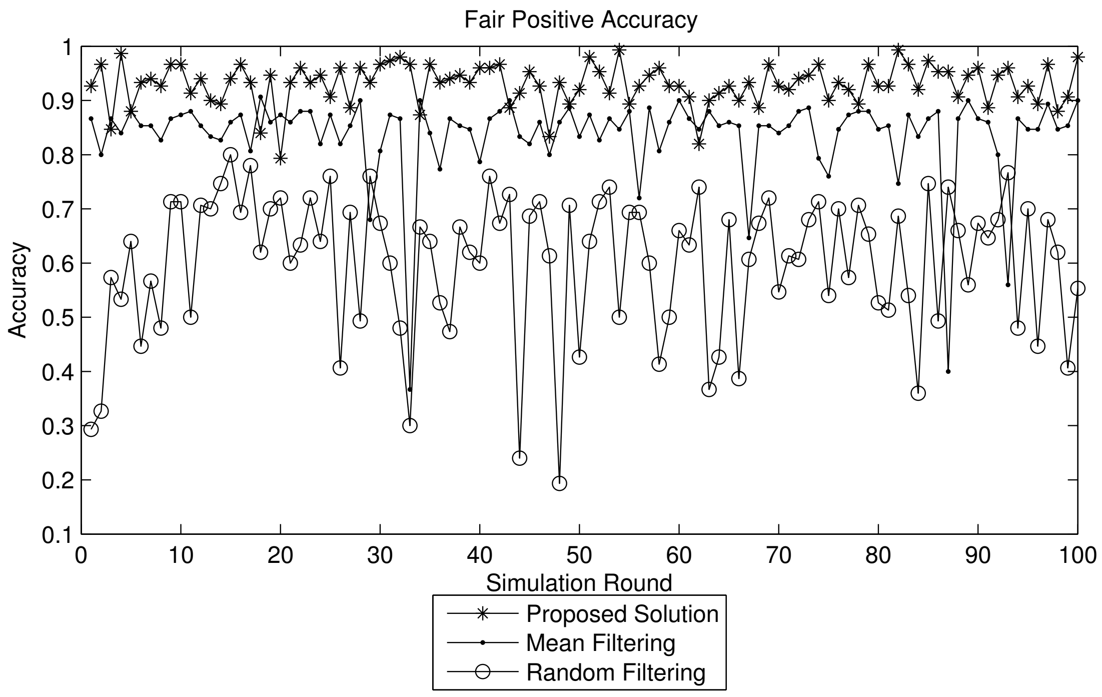


Fig. 6.1. *Accuracy of FP users*

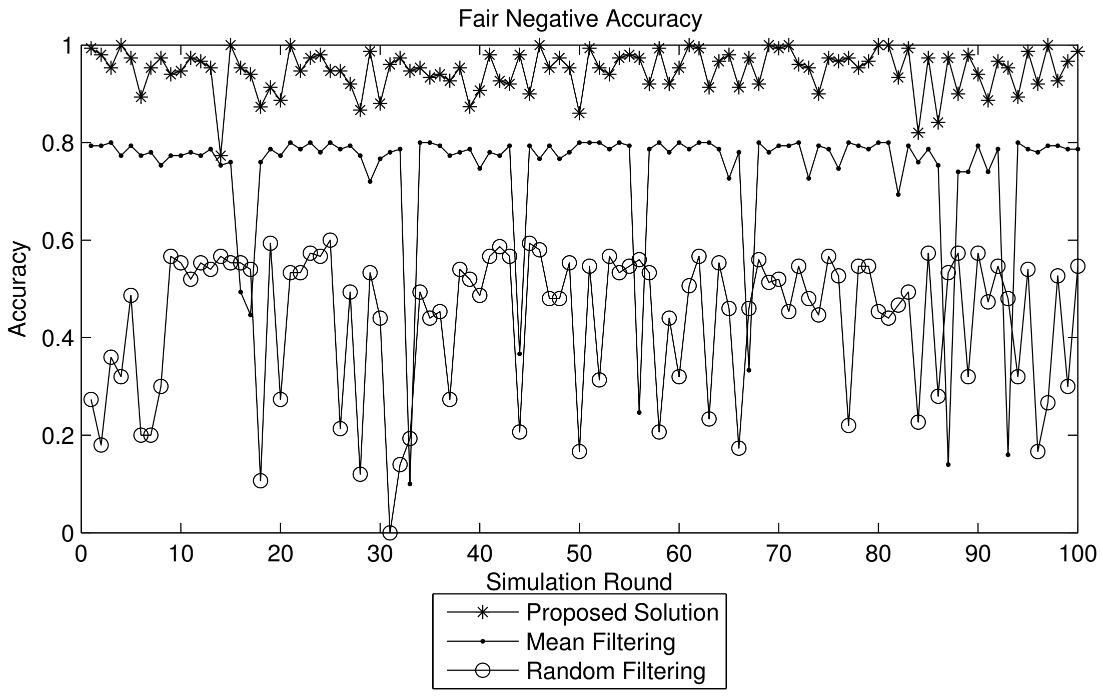


Fig. 6.2. *Accuracy of FN users*

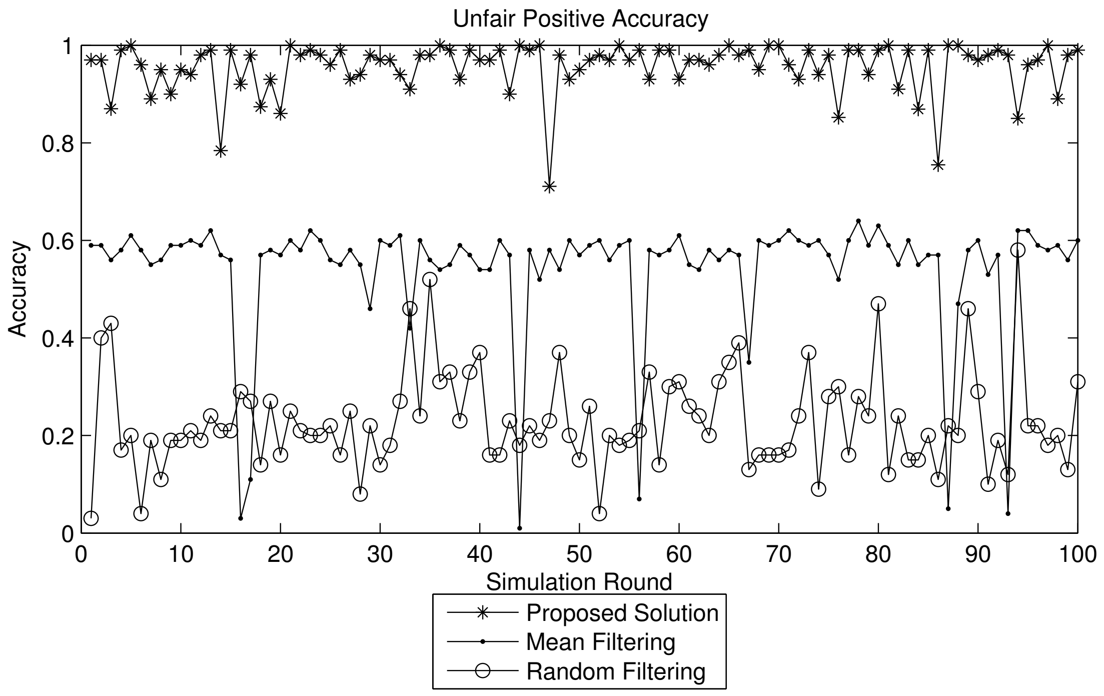


Fig. 6.3. *Accuracy of UP users*

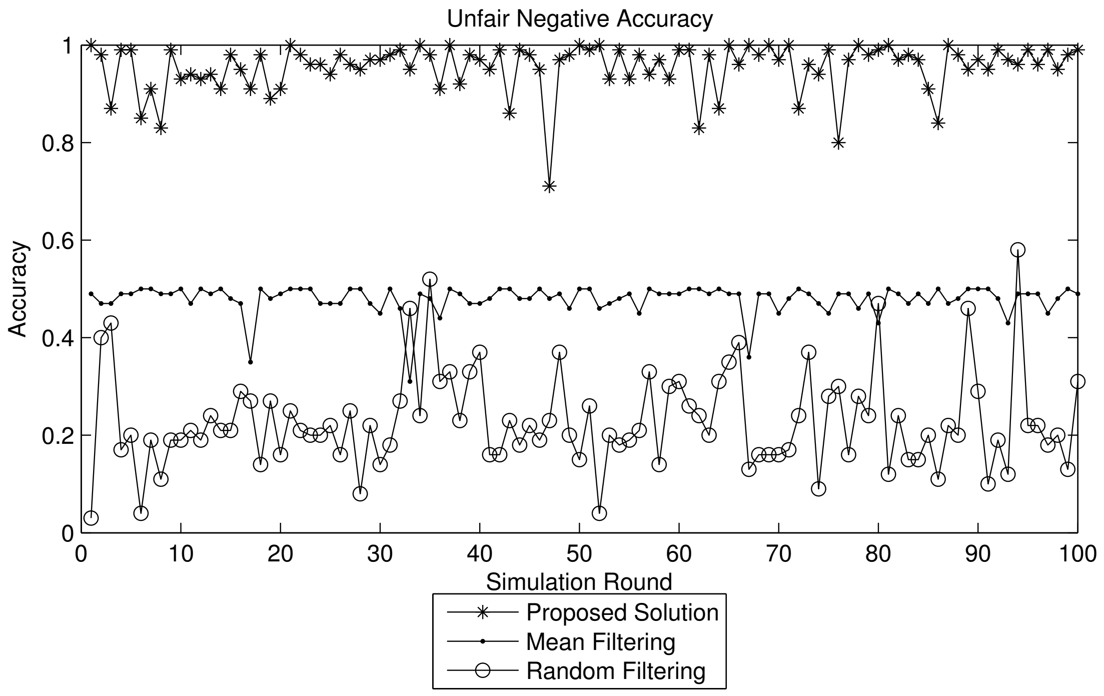


Fig. 6.4. *Accuracy of UN users*

Table 6.3
Accuracy of filtering techniques

Cases	Metrics	Proposed Filtering	Random Filtering	Mean Filtering
Best	MAE	**0.162**	0.365	0.190
	RMSE	**0.220**	0.419	0.311
Average	MAE	**0.181**	0.383	0.239
	RMSE	**0.245**	0.464	0.398
Worst	MAE	**0.193**	0.498	0.348
	RMSE	**0.297**	0.586	0.41

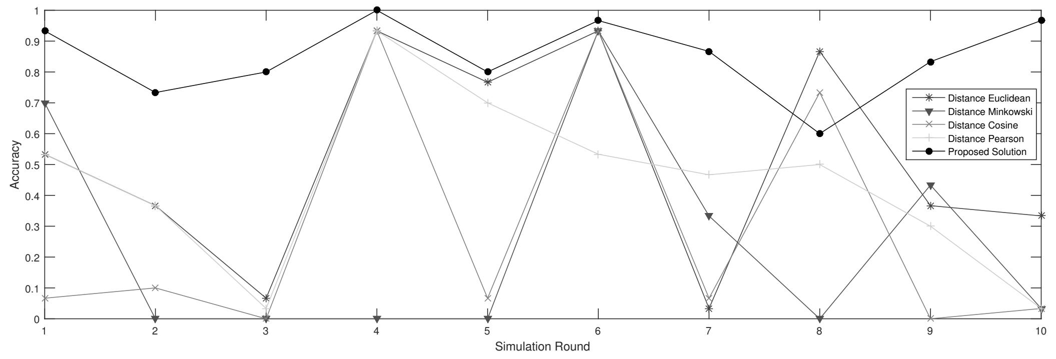


Fig. 6.5. *Accuracy of FP users*

6.3.2. Experiment 2. The results reported in this section were obtained assuming that the total number of users is 500, and the total number of services is 25.

To validate the proposed solution, we have conducted a repeated random sampling for 10 times for the proposed solution with the discussed solutions in section 4 namely Pearson's correlation, Cosine measure, Euclidean distance and Minkowski distance. the experiments were conducted for each measure separately with the same generated sampling.

The experimental results of the accuracy is show in Figure 6.5 for Fair Positive, Figure 6.6 for Fair Negative, Figure 6.7 for Unfair Positive, Figure 6.8 for Unfair Negative. The figure shows that the proposed solution obtained the best classification accuracy representing for all the simulation sounds.

From the results of the five groups of experiments, we can see that our solution obtained the best result in most of the cases while maintaing a definite stability. The result proves that the intersection of the resulted users rating from each measure permit to give better reliability, resulting in a better accuracy of the filtered users.

6.3.3. Experiment 3. In these experiments, we have tested the scalability of the proposed solution regarding the number of users and the number of services.

For the first experiment, we have conducted a repeated random sampling for 10 times with 250 users while increasing the number of the rated services. the experiment was started with 3 services, and increased each round by 5 services, as shown in Figure 6.9.

We can remark from the results that the accuracy of the reported values starts with a very low value which was 20% for the unfair positive group. However, the accuracy increases each round to achieve 100% for services between 23 and 28. This is due to the fact that the low number of services can't give a whole overview of the given rating. Then, with each round the accuracy increases and reaches an average of 80%-90% for services between 8 and 13 which remains very reliable and realistic situations.

The second experiment consists of a repeated random sampling for 10 times with 10 services while increasing

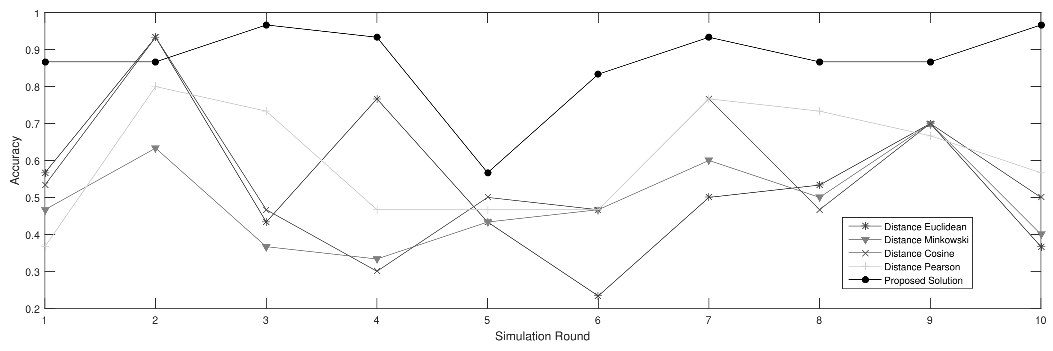


FIG. 6.6. *Accuracy of FN users*

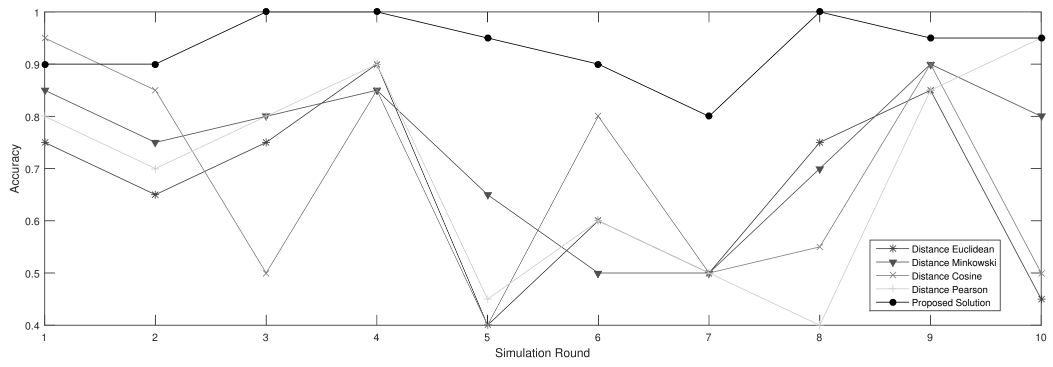


FIG. 6.7. *Accuracy of UP users*

the number of the user ratings. the experiment was started with 200 users, and increased each round by 50 users, as shown in 6.10. We can notice from the results that the accuracy remains stable for each group, which lies between 80% in the worst cases and 100% for the best cases. Hence, we can conclude that the increasing or decreasing of the users number doesn't alter the proposed solution and the results remain very accurate.

Consequently, we can conclude that the experimental results have demonstrated that the proposed technique significantly increase the predicting of the missing values for users rating, to proceed to an accurate clustering of the different groups. This is due to the following reasons. First, in our solution, take into account several measures of distance and similarities comprehensively. Second, the filtering technique would be an appropriate start to compute a reliable trust degree and assess the quality of the proposed services since the clustering is performed without any knowledge about the offered quality of service. Third, the integration of a module to reward or punish the different groups would result in a more accurate trust degree for a trust management system, likewise for the auto-adjusting classification.

7. Conclusion and Future Direction. To protect clouds, traditional security techniques such as encryption and authorization provide a solid basis, but they are insufficient when entities act maliciously over reputations and trust of different service. Trust as a security approach can fight against such threats by restricting malicious entities from participating in interactions and consequently offers a high trustworthiness cloud computing environment. If feedback and rating integrity concerns in trust management and service selection are not addressed sufficiently, the consequences may be severe.

This paper presents a novel classifying model that uses the rating provided by different users by grouping them into four different groups to filter the discussed user attacks. This model can be integrated into a trust management framework to obtain a more specific trust value of the same concept of services. As a future work,

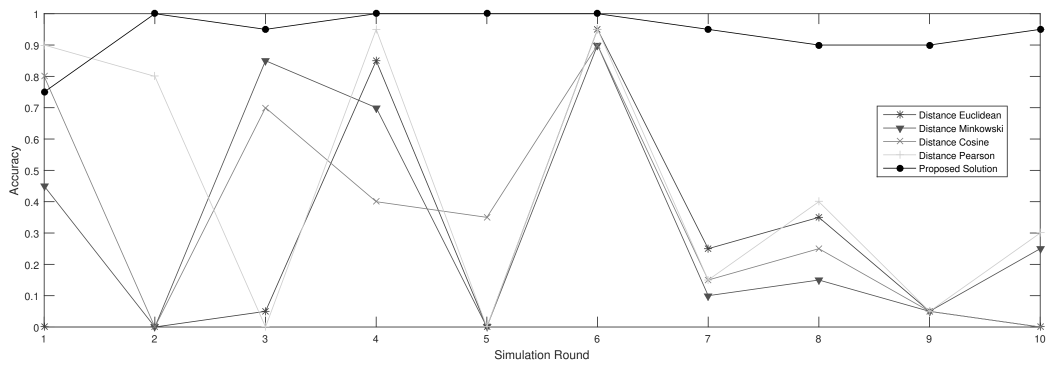


FIG. 6.8. *Accuracy of UN users*

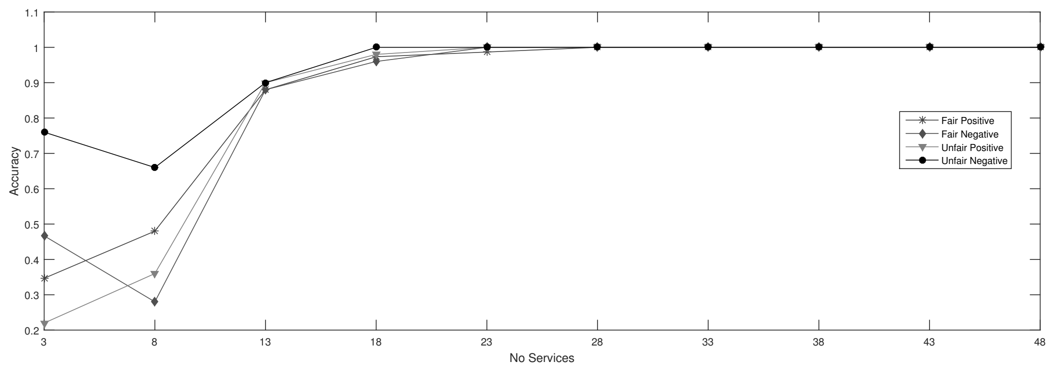


FIG. 6.9. *Services Scalability*

we will discuss design requirements and integration in a trust management system, as well as integrating an authentication service to deal with Sybil attack.

REFERENCES

[1] D. Puthal, B. P. S. Sahoo, S. Mishra, and S. Swain, *Cloud Computing Features, Issues and Challenges: A Big Picture*, no. Cine, 2015.

[2] F. Z. Filali, B. Yagoubi, *Global Trust: A Trust Model for Cloud Service Selection*, International Journal of Computer Network and Information Security, vol. 7, pp. 41-50, 2015.

[3] F. Z. Filali, B. Yagoubi, *A General Trust Management Framework for Provider Selection in Cloud Environment*, Unpublished conference paper at: 19th East European Conference on Advances in Databases and Information Systems (ADBIS 2015), September 8-11, 2015, Poitiers in France.

[4] G. Zhao, C. Rong, M. G. Jaatun, and F. E. Sandnes, *Deployment models: Towards eliminating security concerns from cloud computing*, Proc. 2010 Int. Conf. High Perform. Comput. Simulation, HPCS 2010, pp. 189-195, 2010.

[5] M. Bamiah and S. Brohi, *Seven deadly threats and vulnerabilities in cloud computing*, Int. J. Adv. Eng. Sci. & Techs., no. 9, pp. 87-90, 2011.

[6] Y. L. Sun, Z. Han, W. Yu, and K. J. R. Liu, *A trust evaluation framework in distributed networks: Vulnerability analysis and defense against attacks*, Proc. IEEE INFOCOM 2006. 25TH IEEE Int. Conf. Comput. Commun., pp. 1-13, 2006.

[7] T. H. Noor, Q. Z. Sheng, and A. Alfazi, *Reputation attacks detection for effective trust assessment among cloud services*, Proc. - 12th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. Trust. 2013, pp. 469-476, 2013.

[8] W. Fan and H. Perros, *A reliability-based trust management mechanism for cloud services*, Proc. - 12th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. Trust. 2013, pp. 1581-1586, 2013.

[9] A. Josang and J. Golbeck, *Challenges for robust trust and reputation systems*, 5th Int. Work. 5th Int. Workshop on Security and Trust Management, pp. 1-12, 2009.

[10] K. Thirunarayan, P. Anantharam, C. Henson, and A. Sheth, *Comparative trust management with applications: Bayesian*

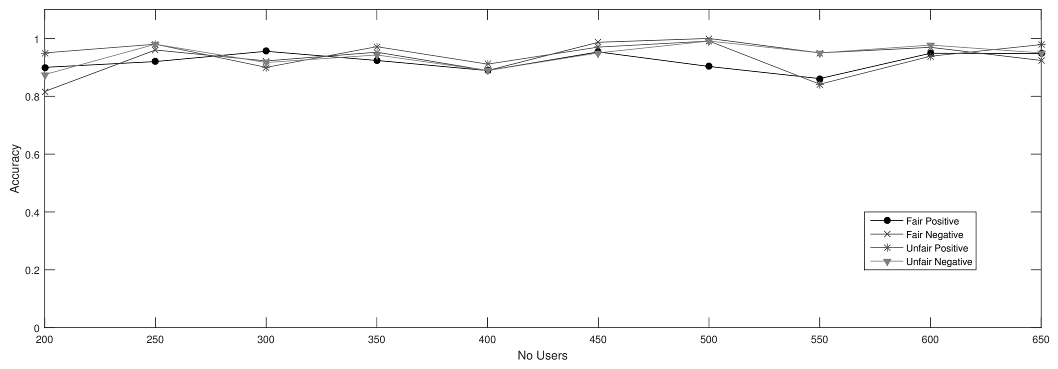


Fig. 6.10. *Users Scalability*

approaches emphasis, Futur. Gener. Comput. Syst., vol. 31, pp. 182-199, 2014.

[11] M. Tavakolifard and K. C. Almeroth, *A Taxonomy to Express Open Challenges in Trust and Reputation Systems*, J. Commun., vol. 7, no. 7, pp. 538-551, Jul. 2012.

[12] J. Trevathan and W. Read, *A simple shill bidding agent*, Proc. - Int. Conf. Inf. Technol. Gener. ITNG 2007, pp. 766-771, 2007.

[13] R. Kerr and R. Cohen, *Smart Cheaters Do Prosper: Defeating Trust and Reputation Systems*, Scenario, pp. 993-1000, 2009.

[14] A. Whitby, A. Josang, and J. Indulska, *Filtering out unfair ratings in bayesian reputation systems*, Chem. Biodivers., vol. 1, no. July, pp. 1829-1841, 2005.

[15] A. Josang and W. Quattrociocchi, *Advanced features in Bayesian reputation systems*, Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 5695 LNCS, pp. 105-114, 2009.

[16] B. Yu and M. P. Singh, *Detecting Deception in Reputation Management*, Proc. Second Int. Jt. Conf. Auton. agents & multiagent Syst., pp. 73-80, 2003.

[17] F. G. Marmol and G. M. Perez, *Security threats scenarios in trust and reputation models for distributed systems*, Comput. Secur., vol. 28, no. 7, pp. 545-556, Oct. 2009.

[18] H. J. Ahn, *A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem*, Inf. Sci. (Ny)., vol. 178, pp. 37-51, 2008.

[19] D. Chrysanthos, *Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior*, Proc. 2nd ACM Conf. Electron. Commer., pp. 150-157, 2000.

[20] M. Chen and J. P. Singh, *Computing and using reputations for internet ratings*, Proc. 3rd ACM Conf. Electron. Commer. - EC '01, pp. 154-162, 2001.

[21] R. Kerr and R. Cohen, *Modeling Trust Using Transactional , Numerical Units*, PST 06 Proc. 2006 Int. Conf. Priv. Secur. Trust Bridg. Gap Between PST Technol. Bus. Serv., pp. 1-11, 2006.

[22] F. G. Marmol, C. Sorge, O. Ugus, and G. M. Perez, *WSANRep, WSAN reputation-based selection in open environments*, Wirel. Pers. Commun., vol. 68, pp. 921-937, 2013.

[23] J. S. Lee, C. H. Jun, J. Lee, and S. Kim, *Classification-based collaborative filtering using market basket data*, Expert Syst. Appl., vol. 29, pp. 700-704, 2005.

[24] S. Gong, *An efficient collaborative recommendation algorithm based on item clustering*, Lect. Notes Electr. Eng., vol. 72 LNEE, pp. 381-387, 2010.

[25] J. G. J. Gu, J. Z. J. Zhou, and X. C. X. Chen, *An Enhancement of K-means Clustering Algorithm*, 2009 Int. Conf. Bus. Intell. Financ. Eng., vol. 2, no. 2, pp. 2-5, 2009.

[26] G. Lekakos and G. M. Giaglis, *A hybrid approach for improving predictive accuracy of collaborative filtering algorithms*, User Model. User-adapt. Interact., vol. 17, pp. 5-40, 2007.

[27] Y. Yao, H. Tong, X. Yan, F. Xu, and J. Lu, *MATRI: a multi-aspect and transitive trust inference model*, Proc. 22nd Int. Conference on World Wide Web, pp. 1467-1476, 2013.

Edited by: Dana Petcu
Received: Apr 9, 2015
Accepted: Aug 15, 2015

DOI 10.12694/scpe.v16i3.1103
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 16, Number 3, pp. 303–319. http://www.scpe.org © 2015 SCPE

EARLY PREDICTION OF THE COST OF CLOUD USAGE FOR HPC APPLICATIONS[*]

MASSIMILIANO RAK[†], MAURO TURTUR, AND UMBERTO VILLANO[‡]

Abstract. After a decade of diffusion, cloud computing has received wide acceptance, but it is not yet attractive for the HPC community. Clouds could be a cost-effective alternative to clusters and supercomputers, providing economy of scale, elasticity, flexibility, and easy customization. Unfortunately, most clouds are optimized for running business applications, not for HPC. However, they can be profitably used to run small-scale parallelism codes.

This paper presents a framework built on the top of a cloud-aware programming platform (mOSAIC) for the development of bag-of-tasks scientific applications. The framework integrates a cloud-based simulation environment able to predict the behavior of the developed applications. Simulations enable the developer to predict at an early development stage performance and cloud resource usage, and so the infrastructure lease cost on a public cloud.

The paper sketches the framework organization and presents the approach followed for the performance simulation of applications, focusing on a software development methodology that hinges on early performance prediction. After showing the results of some validation tests of simulation accuracy, an example of early performance prediction is presented.

Key words: cloud computing, HPC, performance prediction, simulation

AMS subject classifications. 68M14, 68W15

1. Introduction.

After about a decade of fast and widespread diffusion, cloud computing has received wide acceptance, standing as the new information technology platform. However, it is a fact that cloud computing is not yet attractive for the HPC (High Performance Computing) community, leaving the great potential of this paradigm partly underutilized.

At least in theory, clouds could be profitably used to bring the power of economic and scalable parallel computing to the masses, as a cost-effective alternative to clusters and supercomputers. IaaS (Infrastructure as a Service) clouds easily enable users to lease a set of nodes and set up a virtual cluster. Cloud advantages include economy of scale, elasticity, flexibility, and easy customization by exploiting the virtualization of cloud resources. In particular, HPC on clouds appears to be particularly appealing for the users who cannot afford to buy dedicated HPC infrastructures, or have only sporadic parallel computing demands.

However, most clouds (whether public or private) are optimized for running business applications, not for HPC. The biggest obstacles for efficient execution of HPC applications in clouds are the performance losses due to the systematic use of virtualization and, above all, to the use of networks designed mainly for scalability, and not for performance [5]. Moreover, multitenancy, the presence of loads hidden from the user view and control and HPC-agnostic cloud schedulers induce substantial variability of performance from one run to another [20], precluding the possibility to perform fine-grained software optimization. Studies and performance measurements made in the last years [26, 31, 40, 50] have diffused the pessimistic opinion that present-day clouds are simply unfit for the majority of HPC applications.

Simply stated, the problem is that today HPC and clouds are agnostic about each other [23]. There are essentially two options: or i) to re-design both clouds and HPC applications, so as to enable efficient execution of HPC codes in the cloud [24] , or ii) to deploy on the cloud only applications that are relatively unaffected by all the above-mentioned performance inhibitors. These applications include most small-scale parallelism codes, which can be run effectively on tiny virtual clusters, as well as embarrassingly parallel and tree-structured computations, which can exploit a high number of virtual processors without significant performance inefficiency.

This paper moves a step in the second direction. There is a wide range of applications widely used in science, engineering and for commercial purposes that have highly variable response times, are moderately CPU

[*]This paper is an extended version of the paper "Early Prediction of the Cost of HPC Application Execution in the Cloud", presented at the MICAS 2014 Workshop, 22-23 September 2014, West University of Timisoara, Romania.

[†]Department of Industrial and Information Engineering, Second University of Naples, Aversa, Italia, (massiliano.rak@unina2.it)

[‡]Department of Engineering, University of Sannio, Benevento, Italia, (mauro.turtur@unisannio.it, villano@unisannio.it).

intensive, are not fit (unless heavily redesigned) for GPU computing and are made up of loosely coupled tasks, so that computation easily dwarfs communication times. We think that this class of "para-scientific" applications is an almost ideal candidate for execution on the cloud. The major advantage is economic: the cost for leasing a small set of virtual cores can be very low, especially if there are relaxed time constraints for obtaining the results. A wise choice among provider offerings often allows to acquire the computing resources needed at very low cost (see for example the EC2 Spot Instances offer [2]). This enables any organization to run parallel code whenever needed, at a low cost, without investing the capital in rapidly obsolescing parallel hardware. The second important issue is cloud elasticity, which allows to scale in/out the number of virtual cores *on-the-fly* (i.e., while the application is running), based on the particular job requirements, paying just for the resources actually used. In other words, cloud computing is *also* a great opportunity for everyone to experiment and to exploit parallel computing at low cost, using a comfortable pay-as-you-go model.

We think that the final step to make clouds fully advantageous for sporadic scientific users is providing simple tools to predict the performance behavior of their application, allowing them to make a tradeoff between performance and leasing costs. In a previous paper [15] we proposed the use of a cloud-enabled programming platform. This platform makes it possible to develop cloud applications on the top of a cloud-aware programming framework (mOSAIC [42, 43]) by exploiting the bag-of-tasks programming paradigm. The bag-of-tasks (BOT) paradigm, also known as master-worker, processor farm, etc., is widely understood, and ubiquitous in small and medium-scale scientific computing.

Moreover, in the past we worked on the performance prediction of cloud applications developed on the top of the mOSAIC framework [14]. Already-available tools enable us to predict the performance of a cloud application without running it on (payed) cloud resources. In this paper, we discuss the enrichment of the bag-of-task framework with performance prediction capabilities, allowing the automatic generation of the application simulation models.

The remainder of this paper is structured as follows. In the next section we will examine related work. Section 3 illustrates the rationale and the architecture of the framework we have implemented for the development of bag-of-tasks applications in the cloud. Section 4 presents our approach to early performance prediction and Section 6 shows some of our simulation validation tests. Then an example of the use of early prediction to estimate response times and associated cloud leasing costs for a huge-sized problem is presented in Section 7. The paper closes with our conclusions and plans for future research.

2. Related Work.

2.1. HPC in the Cloud. The potential of clouds for scientific computing linked to economicity and to on-demand provision is discussed in [33]. As mentioned in the introduction, in the last few years there have been many works studying the suitability of clouds for HPC computations [11, 18, 27, 28, 29, 30, 36, 37, 50], almost all pointing out the performance losses of at least one of order of magnitude as compared to HPC clusters and the extreme variability of response times. The use of clouds only for particular classes of HPC loads is suggested in [26, 31]. Reference [23] and companion papers [24, 25, 26] suggest the design of HPC-aware clouds. The seminal paper dealing with cloud benchmarking is [49]. More recent results, showing the advances in performance of commercial clouds in the last few years, can be found in [20, 40]. Paper [47] tackles the interesting problem of the optimal use of the low-cost AWS *spot instances*.

In [35], the applicability of cloud platforms, and in particular of Microsoft Azure, to scientific computing is studied by implementing a well-known bioinformatics algorithm (BLAST). An implementation of BOT similar to the one presented in this paper, although with a few significant differences, is presented in [4]. A Java framework for the development of fault-tolerant applications is proposed in [38].

A few papers discuss how to exploit the intrinsic elasticity of clouds, i.e., the ability to increase or decrease the amount of computing resources used for application execution. In [19], the Authors present Cloudine, a platform for the development of generic scientific applications able to exploit at best cloud elasticity. The paper [45] tackles the problem of adding elasticity to existing MPI codes. This is obtained by terminating the execution and restarting the program on a different amount of resources, scaling up/down the number of computing nodes used. The execution of MPI codes over a cloud-aware communication library is discussed in [21], where CMPI, a novel MPI library based on the cloud-oriented communication optimizations proposed in [22], is presented.

2.2. Simulation and Performance Prediction. The core of our proposal is the use of a simulation-based approach for application performance prediction. The use of simulation in the HPC context is widely discussed in a number of papers, as [7, 17]. Recent efforts in this field are documented in [3] and [10].

For Component-Based Software Systems (CBSS), most approaches available in the literature focus on integrating performance prediction and evaluation techniques at design time, i.e., when the system implementation is not available. An exception, which has some similarities with our approach, is the COMPAS system [39]. The paper [32] presents a complete survey of performance strategies for CBSS.

CloudCMP [34] is one of few examples of simulation-based performance predictions of cloud applications, where the goal is to compare different cloud providers and to select a suitable one. Another notable work in this area is CloudSim [9], which targets the simulation of the entire stack of software/hardware components in a cloud infrastructure.

3. The Framework for Science Applications. The solution we proposed in [15] for BOT cloud application development requires several assumptions that span the various steps of the scientific application life-cycle:

- **Development**: the developer of the application provides only the basic sequential code blocks implementing the chosen algorithm. He should not care about communication/synchronization details, but only take into account how data are organized and elaborated.
- **Deployment**: the developer/user should be able to start the application over the cloud, choosing the amount of resources to be used and possibly scaling dynamically them up/down at run time. Fault tolerance is guaranteed by the development framework, and is completely hidden at code level.
- **Execution**: the developer/user submits multiple job to the application, which performs always the same actions over different data.

Our BOT development framework implements the simple and common *split-work-merge* solution pattern. A problem to be solved is split in sub-problems (*tasks*), and handed out to task solvers (*workers*), whose partial results are finally merged (see Figure 3.1). The resulting workflow could be applied also in the context of the map-reduce programming model [16], which involves a similar split-work-merge sequence. The difference is in the timing, as the workers in a bag-of-tasks are not constrained to proceed in lock-step, and can work on subjobs asynchronously among them. Bag-of-tasks applications can be developed by extending the above described components with problem specific-algorithms. The details of the development framework are presented in [15].

The BOT development framework provides all the needed components (*splitter*, *merger*, *worker* and *orchestrator*) and an API that can be used to integrate the user-supplied application code. In fact, all the supplied components are mOSAIC components. mOSAIC is a cloudware that builds up a Platform-as-a-Service on the top of computing resources leased in Infrastructure-as-a-Service mode from a single or even multiple cloud providers [42]. The mOSAIC platform offers an easy way to package and to deploy automatically in the cloud software components. Through the platform interface it is possible to deploy multiple instances of the same component and to restart them, in the case of a failure.

mOSAIC offers a set of already-developed basic components, which can be easily extended by the developer (as we have done in the case of the BOT framework). Among the standard components, `Queues` and `KVstores` play the most important roles.

The mOSAIC `Queue` component is a customized version of the RabbitMQ queue server [48]. It is a software component that offers an API to create messages queues, which can used by the applications to communicate each other. After that a queue has been created, all connected applications can send a message through it. All applications registered as consumers will be able to receive the message.

The `KVstore` component is based on Riak [1]. It offers a persistent NoSQL storage service to cloud applications. Computation and communication components can store data in the shared KVstore components, and retrieve them by a key. For example, the HTTPgw can use the KVstore to store HTTP messages that have a large body; "computing" mOSAIC components (*cloudlets*) can store in a KVstore the results of their elaboration, in order to make them be accessible to the external interface.

4. Performance prediction of bag-of-tasks applications. In the previous section we have described our bag-of-tasks framework for the development of scientific applications in the cloud. A key point is that such applications are fully defined by the number of instances of the mOSAIC components mentioned above and by

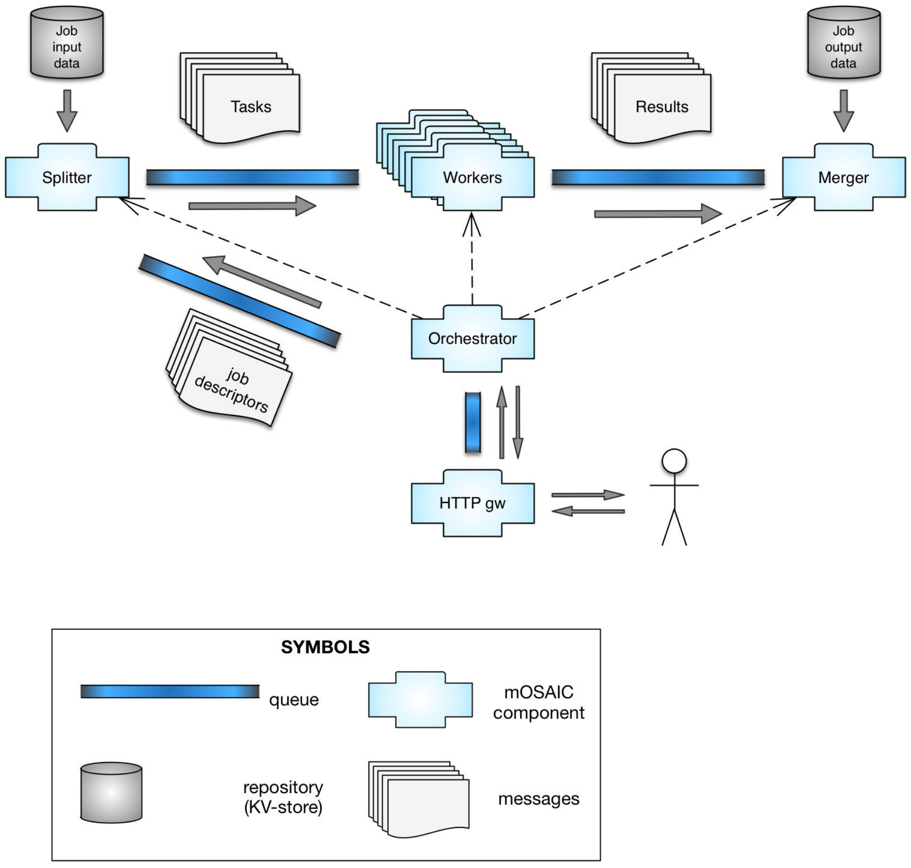


FIG. 3.1. *Architecture of the BOT framework*

their interconnections. Moving from these premises, we have devised a performance model of the application that can be used to predict its behavior and to tune its performance.

Our performance model is process-based [12, 41, 46], in that it is described through a set of discrete-event simulation components whose temporal behavior is described as a process. Event management and discrete-event actions/reactions are modeled in terms of process synchronization primitives. The simulated components have been developed by exploiting the JADES simulation library [12], which allows the description of process-oriented simulations in Java.

A noteworthy feature of the solution devised is that the simulation models expressed through the JADES library can be easily evaluated through the mJADES platform [13]. mJADES is a recently-developed system that supports the distribution of multiple JADES simulations on cloud resources. The mJADES simulation system is based on a Java-based modular architecture. The mJADES *simulation manager* produces simulation tasks from simulation jobs, and schedules them to be executed concurrently on multiple instances of the *simulation core*. This is a process-oriented discrete-event simulation engine based on the JADES simulation library. The outputs from the runs are handed on to a *simulation analyzer*, whose task is to compute aggregates and to generate reports for the final user. mJADES has been developed as a collection of mOSAIC components, and so the evaluation of the models can be performed on any mOSAIC platform. This could be the one where the application under study is deployed and executed, or even a different one.

4.1. Bag-of-tasks Simulation Model. To model a bag-of tasks application, we developed a simulation component for each of the core components making up the bag-of-task framework (`Splitter`, `Worker` and `Merger`), and for each component devoted to manage the cloud application execution (`HTTPgw`, `Orchestrator`),

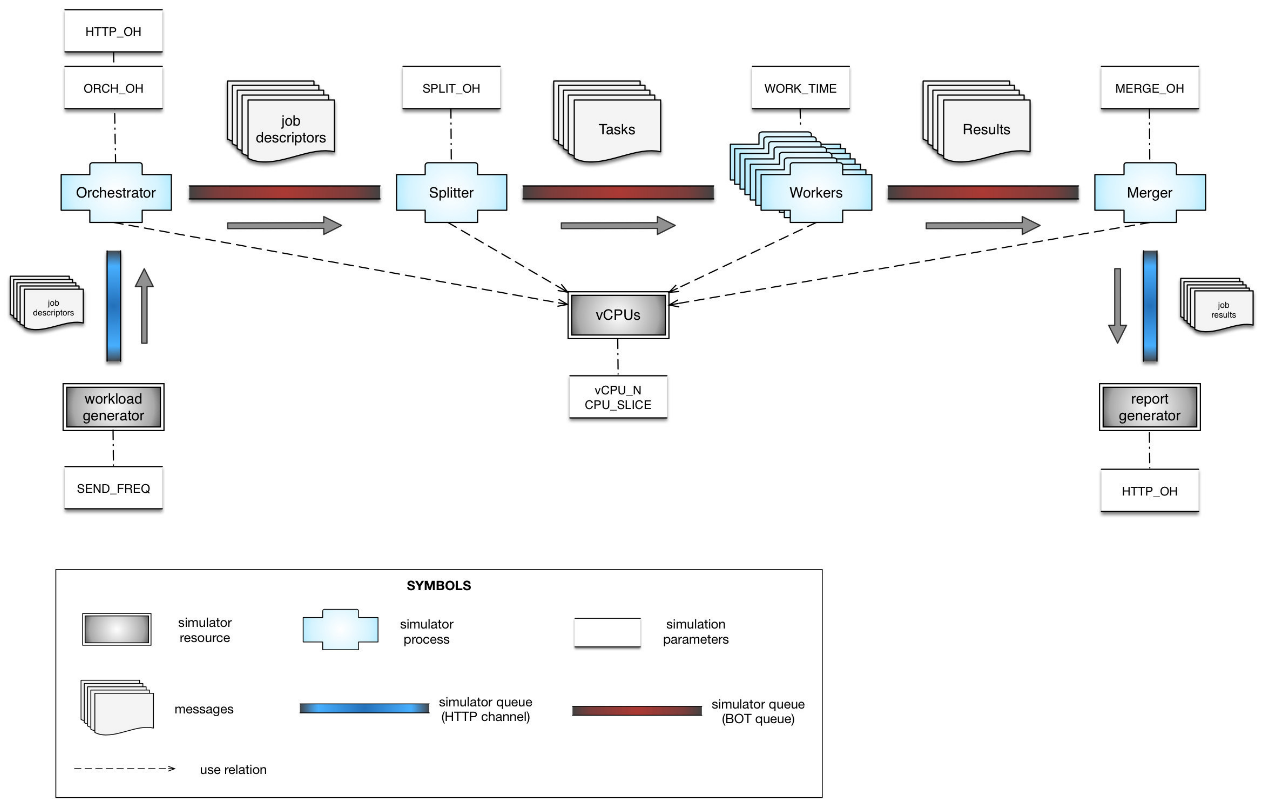


FIG. 4.1. *Bag-Of-Tasks simulation model*

communication and storage (`Queue` and `KVstore`) (Figure 4.1).

The management components of the bag-of-task framework (`HTTPgw` and `Orchestrator`) have the role of managing the cloud application execution, offering an interface to end users (`HTTPgw`) and orchestrating the execution, forwarding the messages to the right splitter when multiple bag-of-tasks are executed on the same resources and starting/stopping the triple Splitter-Worker-Merger.

Our simulation model is driven by a `workload generator`, which is in charge to generate the sequence of requests the users issue in time. At the start of simulation, it begins sending out the messages to the `Orchestrator` according to the chosen workload, described in a configuration file. Two simple workload models are currently available: a set of requests with fixed inter-arrival time, and a Poisson arrival model that generates messages with random exponential inter-arrival time. Moreover, it is also possible to start a multiple number of concurrent workloads miming the load generated by multiple users.

The `Orchestrator` model is fairly simple: it continuously receives `Jobs` from a queue (that mimics an HTTP channel) and forwards them to the `Splitter`. At the state of the art, our model is very simple, since it is based on the assumption of a fixed application configuration, which cannot be dynamically altered (i.e., we cannot start a new set of Split-Work-Merge components during the simulation). So the `Orchestrator` has just the role of routing the message to the `Splitter` instances. We aim at improving the `Orchestrator` model in the future.

The `Splitter`, `Merger` and `Worker` models behave in a similar way, receiving and forwarding the messages from/to the internal queues accordingly to the described bag-of-tasks pattern. The `Merger` process, after collecting all the intermediate job `Result` messages, sends a `job Result` message to a special simulator component, the `report generator`, which gathers the results and produces the final simulation reports.

The computational resources consumed by our core components (i.e., `Orchestrator`, `Splitter`, `Merger`

Listing 1. *The Queue Simulation Process*

```
public class Queue extends it.unisannio.ing.perflab.jades.core.Process {
    private Mailbox inputMailbox;
    private Mailbox outputMailbox;

    public Queue(String name, double beta0, double beta1, double beta2) {
        super(name);
        inputMailbox = new Mailbox(name + "inputMailbox");
        outputMailbox = new Mailbox(name + "outputMailbox");
    }

    public void send(Object m) {
        inputMailbox.send(m);
    }

    public void run() {
        while (true) {
            int msgSize = (Integer) inputMailbox.receive();
            int msgInQueue = inputMailbox.msg_in_queue()+1;
            hold(beta2 * msgSize + beta1 * msgInQueue + beta0);
            outputMailbox.send(msgSize);
        }
    }

    public Object receive() {
        return outputMailbox.receive();
    }

}
```

and `Worker` processes) are taken into account by means of a component that simulates CPU resource sharing. At simulation start-up it is necessary to provide the actual number of available virtual CPUs (vCPUs) and the allocation to vCPUs of the framework components involved in a run.

The response time of the queues (i.e., the time needed to notify a message to a process, once it has been published on the queue) is modeled as a function of the number of queued messages and of the dimension of the messages (according to the iLDS model [44]). Listing 1 shows the code of the process simulating the behavior of the communication queues.

The simulation model sketched in Figure 4.1 can be coded as illustrated in Listing 2. The proposed listing shows only the code fragment in charge of building the simulation application. For brevity's sake we have replaced the actual parameters used for object creation with the placeholder <`parameters`>. It is worth pointing out how closely the simulation code resembles the structure of the bag-of-tasks application.

As previously pointed out, even if the actual framework behavior strictly depends on the specific algorithm to be implemented, in any case the core bag-of-tasks components receive messages from queues and forward them to other queues, consuming a suitable amount of CPU time. Starting from this consideration, we can fully describe a bag-of-tasks instance by means of two sets of parameters.

The *application instance* parameters represent the values under the control of the framework user. They describe the application and the BOT framework configuration to be simulated, as follows:

- **virtual CPUs** (vCPU_N): vCPUs available to the mOSAIC platform;
- **vCPU time slice** (vCPU_SLICE): vCPU scheduler preemption time;
- **workers** (WORKERS): number of Worker instances;
- **splitters** (SPLITTERS): number of Splitter instances;
- **jobs** (JOBS): number of jobs generated and submitted;

Listing 2. *The Application Model Simulation Process*

```
public void run() {
        FilterCloudlet orchestrator, splitter, merger, workers[];
        double setupTime = 0.0;
        hold (setupTime);

        Queue gwQueue=new Queue("gwQueue", <parameters> );
        Queue fromOrchestratorToSplitter=new Queue("orchToSplit",<parameters>);
        Queue fromSplitterToWorker=new Queue("SplitToWork",<parameters>);
        Queue fromWorkerToMerger=new Queue("workToMerge", <parameters>);

        double orchestratorHoldTime =
                Double.valueOf(myProp.getProperty("receive_from_http_overhead"));
        orchestrator = new FilterCloudlet("orchestrator_cloudlet",
                orchestratorHoldTime, gwQueue, fromOrchestratorToSplitter);
        double splitterHoldTime =
                Double.valueOf(myProp.getProperty("single_split_overhead"));
        splitter = new Splitter("splitter_cloudlet",
                splitterHoldTime, fromOrchestratorToSplitter, fromSplitterToWorker);

        add(gwQueue);
        add(fromOrchestratorToSplitter);
        add(fromSplitterToWorker);
        add(fromWorkerToMerger);
        add(outQUeue);
        add(orchestrator);
        add(splitter);

        int workers_num = Integer.valueOf(myProp.getProperty("workers"));
        workers = new FilterCloudlet[workers_num] ;
        double workerThinkTime =
                Double.valueOf(myProp.getProperty("worker_thinktime"));

        for (int i =0 ; i < workers_num ; i++ ){
                workers[i] = new  FilterCloudlet("worker_"+i,
                        workerThinkTime , fromSplitterToWorker, fromWorkerToMerger);
                add(workers[i]);
        }

        double mergerHoldTime =
                Double.valueOf(myProp.getProperty("single_merge_overhead"));
        merger = new Merger("merger_cloudlet",
                mergerHoldTime, fromWorkerToMerger, outQUeue);
        add(merger);

        //create jobs
        int jobs = Integer.valueOf(myProp.getProperty("jobs"));
        double jobs_frequency =
                Double.parseDouble(myProp.getProperty("send_frequence"));
        int tasks = Integer.valueOf(myProp.getProperty("tasks"));

        for (int i =0 ; i < jobs ; i++ ){
        gwQueue.send(new Message(String.valueOf(i)+"_"+tasks));
                hold(jobs_frequency);
        }
}
```

- **send job frequence** (SEND_FREQ): job send rate;
- **tasks** (TASKS): number of tasks generated by the `Splitter`;
- **worker overhead** (WORK_TIME): estimate of the vCPU time required by a `Worker` to process a `Task`;
- **allocation map**: allocation matrix describing the allocation of the framework processes to the available vCPUs.

The *framework tuning parameters* are used to model a specific framework instance, taking into account the overhead introduced by the framework itself, by the platform and by any underlying software layer. After they have been estimated for a framework instance, they will not vary across different simulation runs. In the following subsection we will describe a methodology for the evaluation of such values. The parameters are:

- **HTTP overhead** (HTTP_OH): the communication delay introduced by an HTTP communication channel;
- **Orchestrator overhead** (ORCH_OH): the orchestrator overhead (vCPU time) introduced to process each submitted job and to forward the descriptor to the `Splitter`;
- **Splitter overhead** (SPLIT_OH): overhead introduced (vCPU time) to execute a single *split* operation, to create and to forward a `Task`;
- **Merger overhead** (MERGE_OH): overhead (vCPU time) introduced to execute a single *merge* operation;
- **Job Descriptor message** (JOB_MSG_SIZE): size of the `Job Descriptor` messages;
- **Result message** (RESULT_MSG_SIZE): size of the `Result` messages;
- **Task message** (TASK_MSG_SIZE): size of the `Task` messages;
- **HTTP channel**: *beta0*, *beta1*, *beta2* parameters (see Listing 1) to model the HTTP channel;
- **bag-of-tasks queues**: *beta0*, *beta1*, *beta2* parameters to model the framework internal queues.

4.2. Tuning of Algorithm-dependent Components. The above presented simulation model is completely independent of the algorithm implemented in the bag-of-tasks application. The specific characterics of the application affect the model only as far as the number and dimension of messages the `Splitter` generates, the `Worker(s)` elaborates and the `Merger` joins together are concerned. Moreover, they affect the amount of CPU time required by each of the above components to perform its own actions.

The first set of information (number and dimension of messages generated) is usually dependent only on the problem data dimension, and so is known beforehand, when the simulation takes place. On the other hand, the CPU time needed for each elaboration needs to be estimated. Moreover, the times spent in the other bag-of-tasks framework components (queues, storages, orchestrator) need to be evaluated taking into account the type and number of resources leased from the cloud that will be used to execute the application.

In order to estimate the time behavior of the above mentioned components we use an approach based on the execution of *ad-hoc* benchmarks, which run using the benchmarking framework integrated in mOSAIC API [8]. The methodology used is thoroughly dealt with in paper [14]. Following this approach, we develop dedicated benchmark application for each specific component of the application; the mOSAIC framework automates the process of their execution. Benchmarks for the basic components, namely queue Servers and KV stores, are offered by the mOSAIC framework out-of-the-box. The corresponding benchmarking micro-applications are organized as shown in Figure 4.2. We have developed similar benchmarking applications for the core components of our bag-of-tasks framework (`Splitter`, `Worker` and `Merger`, `Orchestrator`). The execution of such applications, automated by the framework, makes it possible to collect response times in a single csv file, from which, using regression models, it is possible to find the value of the timing parameters to be used for simulation.

5. Early Prediction Development Methodology. Program performance simulation is an interesting matter, but it becomes a fundamental technique if it can be adopted at the very early development stages, before the complete development and deployment of an actual application. The use of performance prediction as an integral part of a parallel program development methodology (*performance debugging*) has turned out to be particularly efficient if performance prediction tools are used at an early stage of the software life cycle (i.e., before a complete the implementation has been devised) [6, 7]. The idea is to exploit a partially implemented program design to obtain performance data before the complete software implementation. This allows a sort of

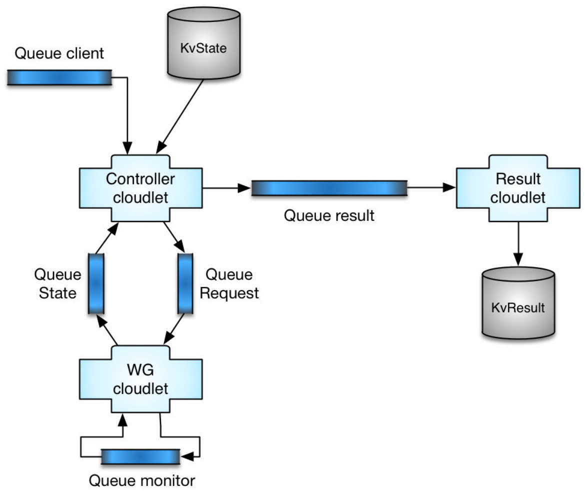


FIG. 4.2. *The Queue Benchmark mOSAIC Application*

performance-driven parallel program design, where algorithm choices can be evaluated and possibly compared very early in the development process.

The "classical" method to obtain early performance data before code finalization is to resort to *prototypes*, i.e., to skeletons of code where some of the computations interleaved between communications are not fully specified, and are represented for simulation purposes by delays equal to the expected time that will be spent in the actual code (for details, please refer to the above cited [6, 7]). In fact, most of the times the specification and the use of code prototypes for simulation purposes is not a simple and error-proof procedure. In the case of BOT code, developed within the proposed framework, instead, the adoption of a "fixed" execution model not only lends itself to a direct construction of a simulation model, as shown in the previous section, but leads to a very simple and well-defined performance prediction development methodology.

This methodology can be directly integrated in the BOT application life cycle (Figure 5.1), and it is made out of the following steps:

1. **Identify the Split/Work/Merge Algorithms**: the developer has to rethink of his/her algorithm in order to identify the role of the core bag-of-tasks components. This is not a complex task, because the BOT paradigm is usually close to the behavior of most common applications. In this phase the developer starts writing and rethinking his/her own code.
2. **Prepare the set of data to be compared**: concurrently with algorithm development, the developer identifies the way in which the application will be used in the future, i.e., how many requests will be served and how the work will be split among workers. This activity can be conducted concurrently with development, so that it is possible to adapt splitting and merging algorithms in order to reduce the execution costs in future, on the basis of the simulation predictions.
3. **Execute the benchmarks and collect the results**: the benchmark applications are launched in the target mOSAIC cloud platform, i.e., in the same environment where the final code will be executed. This makes it possible to obtain performance figures for every component *on the actual execution platform*. It should be noted that, if the application is not fully developed, the developer will only run the benchmarks for the core components. The CPU time requirements of not-fully developed code can be estimated by means of static software analysis. Of course, this is likely to affect adversely the prediction accuracy.
4. **Obtain a performance prediction by executing the simulation model**: the simulation model can be executed with multiple synthetic workloads, using the parameters estimated as described in the previous steps, obtaining performance predictions for different scenarios of interest.
5. **Cost evaluation**: the response times of the application, possibly under several different configuration

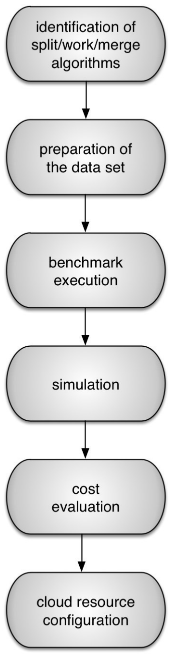

identification of split/work/merge algorithms
preparation of the data set
benchmark execution
simulation
cost evaluation
cloud resource configuration

FIG. 5.1. *The Phases of the Early Prediction Methodology*

options, can be "projected" on the cloud offerings of multiple providers, considering alternative pricing plans. The objective here is to find the application execution cost under any reasonable configuration.

6. **Cloud resource configuration**: the output of the previous step makes it possible for the developer to compare providers, pricing plans and configuration options. Here the developer may choose to trade off performance for reduced cost of leased resources, or to focus on performance at the expense of a higher cost of computing resources.

Following such approach, the developer is able to predict the performance and resource usage (and related costs) from the early development stages of his/her cloud application. It is worth pointing out that the proposed procedure does not require the complete set of cloud resources corresponding to the chosen run-time configuration to find predicted response times. In other words, it is possible to build the model and to obtain by benchmarking the performance figures of every system component on a small-scale virtual machine configuration. Then, it is possible to obtain by simulation the overall application response time on a much larger machine configuration. This option will be discussed in detail in the example presented in Section 7.

6. Validation of Simulation Results.

As outlined in the previous sections, the framework we propose is composed of (i) the cloud BOT development framework, which enables the developer to run applications on cloud environment, (ii) the simulation framework, which enables the developer to predict application performance even in the early development stages and (iii) the benchmark applications, which are used to evaluate the timing parameters for simulation.

The first step needed to evaluate the feasibility of the proposed approach is to validate the (small-scale) performance results obtained by simulation. A more comprehensive performance prediction and analysis example, including cost evaluation and comparisons, will be presented in the next Section.

In order to validate the approach, we run benchmark applications on the target VMs to gather the timing parameters for the simulation environment, using a minimal set of resources. Then we compare a real application with its simulation, varying both the workload and the amount of resources assigned to the application run.

6.1. Measurement of Timing Parameters.

Our BOT framework makes it possible to obtain applications that are completely independent of the real environment on which it will run. The number and the characteristics of cloud resources leased for execution only affect performance. We capture the characteristics of the actual execution environment through a set of benchmarks, whose results produce the timing parameters successively used for simulation.

For our tests, we have deployed the mOSAIC platform on Virtual Machines (VM) leased from the Amazon Web Services (AWS) infrastructure. The characteristics of the acquired VMs are shown in Table 6.1. Running our benchmark suite, we obtained the values in Table 6.2.

Table 6.1
AWS VM instance details

Parameter	Value
instance type	c1.medium
vCPU	2
RAM	1.7 GB
storage	350 GB
network performance	moderate

Table 6.2
Testbed timing parameters

Parameter Name	Value
HTTP_OH	50 ms
ORCH_OH	10 ms
SPLIT_OH	5 ms
MERGE_OH	5 ms
HTTP_CHANNEL beta0	1500
HTTP_CHANNEL beta1	0.500
HTTP_CHANNEL beta2	0.010
BOT_QUEUES beta0	65
BOT_QUEUES beta1	0.003
BOT_QUEUES beta2	0.001

6.2. Simulation Validation Varying the Workload. We have developed a skeletal BOT application, where the work method in the `Worker` class is able to process a given load, expressed in MFLOPS and obtained as a parameter from the task description. The Splitter and Merger do not actually perform splitting/merging, but just activate Workers and collect response times, respectively. We submitted the same workload both to the skeletal application running in the real execution environment and to the simulator, comparing the measured and predicted completion times. For our tests, we used the workloads briefly described in Table 6.3, which are representative of light (WORK_TIME=15 ms) and medium-heavy (WORK_TIME=200 ms) load for the workers. To vary dynamically the number of tasks, we used a pseudo-random uniform distribution.

Figure 6.1 shows on the x-axis the job number (30 jobs are submitted, according to Table 6.3) and on the y-axis its completion time. The completion time associated to job #30 is the total completion time for the whole burst of 30 jobs. The simulation results are summarized in Table 6.4, where is also reported the estimated vCPU usage.

Summarizing the results shown in Figure 6.1, for Test 1 (WORK_TIME=15 ms) we measured a completion time of 512 ms versus a predicted one of 551 ms, with a relative error of 7.61% (considering also intermediate jobs completion times, the error ranges from a minimum of 0.77% to a maximum of 28.57%). For Test 2 (WORK_TIME=200 ms) we measured a completion time of 1383 ms against a predicted of 1134 ms, with a relative error of about 18% (considering intermediate jobs, the error ranges from 4.37% to 18.62%).

In both cases the simulation offers good predictive capacities. Moreover, even at this stage (no actual application developed) it offers interesting information about resource usage. The vCPU usage in Test 1 (the one with lower WORK_TIME) is very low, as most of execution time is spent in communications. This is a

TABLE 6.3
Workload for Test 1 and 2

parameter	value
vCPU	2
JOBS	30
SEND_FREQ	1 s
TASKS	uniform(100,400)
WORK_TIME	15 ms; 200 ms
WORKERS	4
SPLITTERS	2
TASK_MSG_SIZE	400 B
JOB_MSG_SIZE	20 B
RESULT_MSG_SIZE	20 B

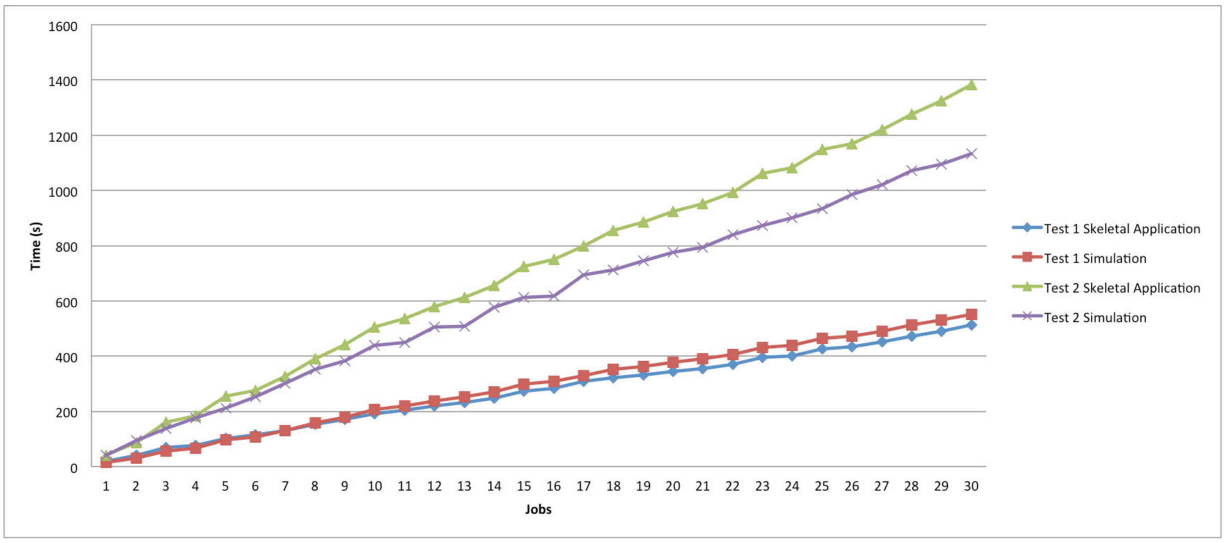


FIG. 6.1. *Comparison between skeletal application and simulated completion times*

relevant hint for the developer of real code, who can tune its implementation so as to obtain a coarser task granularity and hence a more efficient balance of computing and communication.

6.3. Simulation Validation Varying the Resources Used. In the following we compare the performance of the skeletal application executed on real resources to the one obtained by simulation, using a variable number of workers for the real and the simulated run. It should be noted that a higher number of vCPUs is clearly necessary for the execution of these tests on real resources. On the other hand, if we assume to lease additional vCPUs on AWS of the same type used in the previous test, we can use once again the timing parameters in Table 6.2. So the simulation of a larger-scale run does not involve additional leasing costs. Only the allocation matrix in the simulator configuration file has to be updated.

We submitted the workload described in Table 6.5 to the synthetic application, varying the number of workers and comparing it to the real measurements on the skeletal application. The test outcome matches the results proposed in [15], where we evaluated the framework overhead.

Table 6.6 shows the simulation results compared to the actual figures obtained through the real execution of the synthetic application. It should be noted that the simulation output offers the same information on scalability as tests on real hardware (presented previously in [15]). This is a proof of the simulation ability of to catch and to reproduce the behavior of the real system. This ability can be fruitfully exploited to avoid expensive executions on real resources.

Table 6.4
Test simulation results

Metric	Test 1	Test 2
Completion Time	551	1134
Tasks sent	7354	7354
vCPU1 usage	0.29%	0.99%
vCPU2 usage	0.04%	0.99%

Table 6.5
Workload for Worker Scalability Test

parameter	value
vCPU	4
JOBS	10
SEND_FREQ	1 s
TASKS	30
WORK_TIME	100 ms
WORKERS	1 to 5
SPLITTERS	1
TASK_MSG_SIZE	20 B
JOB_MSG_SIZE	20 B
RESULT_MSG_SIZE	20 B

7. Cost/Performance Analysis by Early Prediction.

The goal of this paper is not to provide ultimate performance measurements for real-world applications. Our objective is just to evaluate the feasibility of the proposed approach, i.e., the development of scientific code on the top of a cloud-aware programming framework, exploiting early performance prediction techniques for making cost/performance trade-offs.

As mentioned before, the main advantage of the adoption of simulation is that it enables us to make predictions of performance (and related costs) even when complete code and/or resources are not available. Assuming that the simulation results are sufficiently accurate, as in the example of the previous section, it is possible to test different cloud resource configurations/options at no additional cloud leasing cost. The evaluations thus obtained, as early as at application development time, can help developers to make the choice most fit for their needs.

7.1. Application Configuration Optimization.

The simulator helps to drive application development and deployment, predicting the effect of different configurations and of the amount of computing resources used. It should be noted that resorting to real measurements to support development and configuration choices is inefficient and expensive. Furthermore, real measurements require the availability of (at least) a code skeleton, whose construction is likely to be cumbersome. A possible option to perform predictions and comparisons would be the use of an analytic model of the application. However, the construction of an analytic model requires an in-depth knowledge of the performance behavior of system and application. All things considered, the most simple solution is the use of the general-purpose simulation model presented in Section 4, which makes it is possible to obtain reasonably-accurate evaluations in a simple and straightforward way.

The objective of the following case study is to use performance and cost prediction to support the choice of an optimal number optimal number of vCPU for the workload shown in Table 7.1. This is very different from the workloads of Tables 6.3 and 6.5, since there just one huge job (and not a stream of small successive jobs) made out of many tasks (100,000) with long work time (3000 ms). The investigation is conducted for a number of vCPUs ranging from 1 to 20, assuming to map a worker on each vCPU.

The use of the same AWS instances as in the previous section (see Table 6.1) and hence of the same timing parameters (Table 6.2) made it possible to run all the required simulations on a PC in few minutes, without leasing resources from the cloud. The obtained completion times are shown in Table 7.2. Taking into account

Table 6.6
Scalability: Real and Simulated Response times

Number of Workers	Real App Response Time	Simulated Time (s)
1	84	94
2	44	49
3	31	35
4	30	33
5	30	33

Table 7.1
Case study workload

parameter	value
vCPU	1,2,4,8,16, 20
JOBS	1
SEND_FREQ	N/A
TASKS	100000
WORK_TIME	3000 ms
WORKERS	2,4,8
SPLITTERS	1
TASK_MSG_SIZE	20 B
JOB_MSG_SIZE	20 B
RESULT_MSG_SIZE	20 B

that the C1 medium instances are priced at 0.130/hour[1], it is possible to plot response time and running cost as shown in Figure 7.1. Without entering for brevity's sake into the detail of the comparison between different providers, type of instances and pricing plans, the proposed figure makes it possible to observe that for more of 8 vCPUs the cost rises without any significant performance increase. In these conditions, the most reasonable choice is probably the use of 4 or 8 vCPUs for workers.

The presented trends can be obtained at early application development stage, since the process requires, in essence, only to predict the computational load to process each task. On the contrary, the other parameters represent design choices.

8. Conclusions and Future Work.

The aim of the work described in this paper is to propose an approach that integrates the development of scientific application on top of a cloud platform and its performance prediction through a dedicated simulation environment.

To obtain this integration, on one hand we have built a development framework, currently specialized for the bag-of-task paradigm, which exploits the API and the components provided by the mOSAIC platform. On the other, we have developed a set of simulation components for JADES. These simulation components correspond one-to-one to the mOSAIC components for the BOT framework. Besides presenting the approach, we have discussed the outcome of our preliminary performance tests used to evaluate the simulation accuracy.

In our tests, we offered some examples of usage under different workloads and using different amount of resources, in order to show how it will be possible for a developer to predict the behavior of is application, its completion time and the associated cloud resource leasing cost without running it on (paid) cloud resources, but simply using the associated simulation environment.

Our future research work will focus on the extensive testing of the framework and simulation components, by collecting measurements on real-world scientific codes running in private and commercial cloud environments. We also plan to implement alternative frameworks for additional programming paradigms.

[1]Price checked on 10/6/2015: *on-demand* pricing plan, US East (North Virginia) data center.

TABLE 7.2
Completion times vs. number of workers

# Workers	Completion Time (h)
1	83472,32278
2	41793,99694
4	20885,73444
8	10461,88806
12	7396,314722
16	6503,216667
20	6139,974722

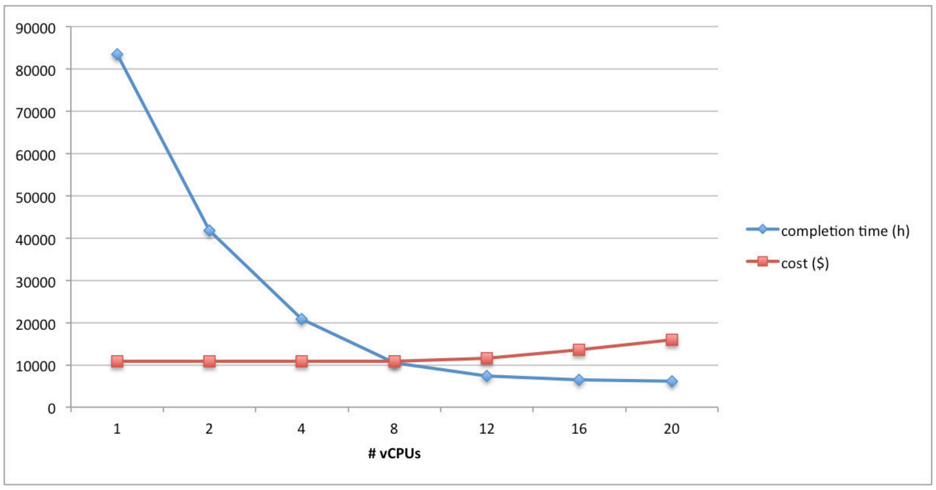


FIG. 7.1. *Completion time and cost vs. number of vCPUs*

REFERENCES

[1] *Basho products.* "http://basho.com/riak/".
[2] *ec2 spot instances.* "https://aws.amazon.com/ec2/purchasing-options/spot instances/".
[3] S. ACHOUR, M. AMMAR, B. KHMILI, AND W. NASRI, *MPI-PERF-SIM: Towards an automatic performance prediction tool of MPI programs on hierarchical clusters*, in Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on, IEEE, 2011, pp. 207–211.
[4] D. AGARWAL AND S. PRASAD, *Azurebot: A framework for bag-of-tasks applications on the azure cloud platform*, in Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International, May 2013, pp. 2139–2146.
[5] M. AL-FARES, A. LOUKISSAS, AND A. VAHDAT, *A scalable, commodity data center network architecture*, SIGCOMM Comput. Commun. Rev., 38 (2008), pp. 63–74.
[6] R. AVERSA, A. MAZZEO, N. MAZZOCCA, AND U. VILLANO, *Developing applications for heterogeneous computing environments using simulation: A case study*, Parallel Computing, 24 (1998), pp. 741 – 761.
[7] R. AVERSA, A. MAZZEO, N. MAZZOCCA, AND U. VILLANO, *Heterogeneous system performance prediction and analysis using ps*, Concurrency, IEEE, 6 (1998), pp. 20–29.
[8] G. AVERSANO, M. RAK, AND U. VILLANO, *The mosaic benchmarking framework: Development and execution of custom cloud benchmarks.*, Scalable Computing: Practice and Experience, 14 (2013).
[9] R. CALHEIROS, R. RANJAN, A. BELOGLAZOV, C. DE ROSE, AND R. BUYYA, *Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms*, Software: Practice and Experience, 41 (2011), pp. 23–50.
[10] P. CLAUSS, M. STILLWELL, S. GENAUD, F. SUTER, H. CASANOVA, AND M. QUINSON, *Single node on-line simulation of MPI applications with SMPI*, in 25th IEEE International Parallel and Distributed Processing Symposium, 2011. IPDPS'11, IEEE, 2011, pp. 664–675.
[11] E. F. COUTINHO, G. PAILLARD, AND J. N. DE SOUZA, *Performance analysis on scientific computing and cloud computing environments*, in Proceedings of the 7th Euro American Conference on Telematics and Information Systems, EATIS '14, New York, NY, USA, 2014, ACM, pp. 5:1–5:6.

[12] A. Cuomo, M. Rak, and U. Villano, *Process-oriented discrete-event simulation in Java with continuations:quantitative performance evaluation*, in Proc. of the International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), SciTePress, 2012, pp. 87–96.

[13] A. Cuomo, M. Rak, and U. Villano, *Concurrent simulation in the cloud with the mjades framework*, International Journal of Simulation and Process Modelling, (2013), pp. 212–226.

[14] A. Cuomo, M. Rak, and U. Villano, *Performance prediction of cloud applications through benchmarking and simulation*, International Journal of Computational Science and Engineering, in the press (2014).

[15] A. De Benedictis, M. Rak, M. Turtur, and U. Villano, *Cloud-aware development of scientific applications*, in 23rd IEEE International WETICE Conference WETICE-2014, June 2014, pp. 149–154.

[16] J. Dean and S. Ghemawat, *Mapreduce: Simplified data processing on large clusters*, Commun. ACM, 51 (2008), pp. 107–113.

[17] B. Di Martino, E. Mancini, M. Rak, R. Torella, and U. Villano, *Cluster systems and simulation: from benchmarking to off-line performance prediction*, Concurrency and Computation: Practice and Experience, 19 (2007), pp. 1549–1562.

[18] C. Evangelinos and C. N. Hill, *Cloud computing for parallel scientific HPC applications: Feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2*, in 1st Workshop on Cloud Computing and its Applications (CCA), 2008.

[19] G. Galante and L. Bona, *Constructing elastic scientific applications using elasticity primitives*, in Computational Science and Its Applications ICCSA 2013, B. Murgante, S. Misra, M. Carlini, C. Torre, H.-Q. Nguyen, D. Taniar, B. Apduhan, and O. Gervasi, eds., vol. 7975 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 281–294.

[20] L. Gillam, B. Li, J. O'Loughlin, and A. P. S. Tomar, *Fair benchmarking for cloud computing systems*, Journal of Cloud Computing: Advances, Systems and Applications, 2 (2013).

[21] Y. Gong, B. He, and J. Zhong, *An overview of CMPI: Network performance aware MPI in the cloud*, in Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '12, New York, NY, USA, 2012, ACM, pp. 297–298.

[22] Y. Gong, B. He, and J. Zhong, *Network performance aware MPI collective communication operations in the cloud*, Parallel and Distributed Systems, IEEE Transactions on, (2013).

[23] A. Gupta, *Efficient high performance computing in the cloud: Keynote talk*, in Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing, VTDC '15, New York, NY, USA, 2015, ACM, pp. 1–1.

[24] A. Gupta and L. Kale, *Towards efficient mapping, scheduling, and execution of HPC applications on platforms in cloud*, in Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International, May 2013, pp. 2294–2297.

[25] A. Gupta, L. Kale, D. Milojicic, P. Faraboschi, and S. Balle, *HPC-aware VM placement in infrastructure clouds*, in Cloud Engineering (IC2E), 2013 IEEE International Conference on, March 2013, pp. 11–20.

[26] A. Gupta and D. Milojicic, *Evaluation of HPC applications on cloud*, in Open Cirrus Summit (OCS), 2011 Sixth, Oct 2011, pp. 22–26.

[27] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, *Case study for running HPC applications in public clouds*, in Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, New York, NY, USA, 2010, ACM, pp. 395–401.

[28] Z. Hill and M. Humphrey, *A quantitative analysis of high performance computing with Amazon's EC2 infrastructure: The death of the local cluster?*, in Grid Computing, 2009 10th IEEE/ACM International Conference on, Oct 2009, pp. 26–33.

[29] A. Iosup, S. Ostermann, M. Yigitbasi, R. Prodan, T. Fahringer, and D. H. J. Epema, *Performance analysis of cloud computing services for many-tasks scientific computing*, Parallel and Distributed Systems, IEEE Transactions on, 22 (2011), pp. 931–945.

[30] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright, *Performance analysis of high performance computing applications on the Amazon web services cloud*, in Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, IEEE, 2010, pp. 159–168.

[31] G. Juve, E. Deelman, G. B. Berriman, B. P. Berman, and P. Maechling, *An evaluation of the cost and performance of scientific workflows on Amazon EC2*, Journal of Grid Computing, 10 (2012), pp. 5–21.

[32] H. Koziolek, *Performance evaluation of component-based software systems: A survey*, Performance Evaluation, 67 (2010), pp. 634–658.

[33] C. A. Lee, *A perspective on scientific cloud computing*, in Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, New York, NY, USA, 2010, ACM, pp. 451–459.

[34] A. Li, X. Yang, S. Kandula, and M. Zhang, *Cloudcmp: comparing public cloud providers*, in Proceedings of the 10th annual conference on Internet measurement, ACM, 2010, pp. 1–14.

[35] W. Lu, J. Jackson, and R. Barga, *Azureblast: A case study of developing science applications on the cloud*, in Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, New York, NY, USA, 2010, ACM, pp. 413–420.

[36] A. Marathe, R. Harris, D. K. Lowenthal, B. R. de Supinski, B. Rountree, M. Schulz, and X. Yuan, *A comparative study of high-performance computing on the cloud*, in Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing, HPDC '13, New York, NY, USA, 2013, ACM, pp. 239–250.

[37] P. Mehrotra, J. Djomehri, S. Heistand, R. Hood, H. Jin, A. Lazanoff, S. Saini, and R. Biswas, *Performance evaluation of Amazon EC2 for NASA HPC applications*, in Proceedings of the 3rd Workshop on Scientific Cloud Computing Date, ScienceCloud '12, New York, NY, USA, 2012, ACM, pp. 41–50.

[38] E. Mocanu, V. Galtier, and N. Tapus, *Generic and fault-tolerant bag-of-tasks framework based on javaspace technology*, in 2012 IEEE International Systems Conference, March 2012, pp. 1–6.

[39] A. Mos and J. Murphy, *A framework for performance monitoring, modelling and prediction of component oriented dis-*

tributed systems, in Proceedings of the 3rd international workshop on Software and performance, ACM, 2002, pp. 235–236.

[40] J. Napper and P. Bientinesi, *Can cloud computing reach the top500?*, in Proceedings of the Combined Workshops on UnConventional High Performance Computing Workshop Plus Memory Access Workshop, UCHPC-MAW '09, New York, NY, USA, 2009, ACM, pp. 17–20.

[41] K. Perumalla and R. Fujimoto, *Efficient large-scale process-oriented parallel simulations*, in Proc. of the 30th Winter Simulation Conference, 1998, pp. 459–466.

[42] D. Petcu, C. Craciun, M. Neagul, S. Panica, B. D. Martino, S. Venticinque, M. Rak, and R. Aversa, *Architecturing a sky computing platform*, in ServiceWave Workshops, M. Cezon and Y. Wolfsthal, eds., vol. 6569 of Lecture Notes in Computer Science, Springer, 2010, pp. 1–13.

[43] D. Petcu and M. Rak, *Open-source cloudware support for the portability of applications using cloud infrastructure services*, in Cloud Computing, Z. Mahmood, ed., Computer Communications and Networks, Springer London, 2013, pp. 323–341.

[44] M. Rak, R. Aversa, B. D. Martino, and A. Sgueglia, *Web services resilience evaluation using lds load dependent server models*, JCM, 5 (2010), pp. 39–49.

[45] A. Raveendran, T. Bicer, and G. Agrawal, *A framework for elastic execution of existing MPI programs*, in Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on, May 2011, pp. 940–947.

[46] H. Schwetman, *CSIM19: a powerful tool for building system models*, in Proc. of the 33nd Winter Simulation Conference, IEEE, 2001, pp. 250–255.

[47] M. Taifi, *Banking on decoupling: Budget-driven sustainability for HPC applications on auction-based clouds*, SIGOPS Oper. Syst. Rev., 47 (2013), pp. 41–50.

[48] A. Videla and J. J. W. Williams, *RabbitMQ in action: distributed messaging for everyone. Rabbit MQ in action*, Manning, Shelter Island NY, 2012.

[49] E. Walker, *Benchmarking Amazon EC2 for high-performance scientific computing*, ; login:: the magazine of USENIX & SAGE, 33 (2008), pp. 18–23.

[50] Y. Zhai, M. Liu, J. Zhai, X. Ma, and W. Chen, *Cloud versus in-house cluster: Evaluating Amazon cluster compute instances for running MPI applications*, in State of the Practice Reports, SC '11, New York, NY, USA, 2011, ACM, pp. 11:1–11:10.

Edited by: Dana Petcu
Received: June 16, 2015
Accepted: Aug 21, 2015

DOI 10.12694/scpe.v16i3.1104
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 16, Number 3, pp. 321–332. http://www.scpe.org © 2015 SCPE

CHALLENGES IN FORMAL METHODS FOR TESTING AND VERIFICATION OF CLOUD COMPUTING SYSTEMS [*]

AMJAD GAWANMEH[†] AND AHMAD ALOMARI[‡]

Abstract. Formal methods are necessary to capture the semantics and behavior of processes of various systems. They characterize and provide insight into the behavior of real systems and thus identify their deterministic and non-deterministic features. The design and deployment of cloud computing systems utilize the current technology development in order to provide the appropriate service and accommodate the increasing demand while maintaining high quality and error free service. In this paper, we discuss the state of the art on using formal methods for the verification of cloud computing systems. Even though formal methods have been used successfully in the design and verification of several aspects of these systems, there are still many design issues in cloud computing that can be enhanced using formal methods. For instance, several scheduling algorithms are being used for cloud frameworks, such as Hadoop for instance, that are found to suffer from scheduling failures. This could have been avoided if the schedular has been properly verified. On the other hand, several new paradigms have evolved with cloud computing such as big data, these require fundamental changed on methods and algorithms that are being used for classical distributed systems, which in turn, increase the chance of having faulty systems that are difficult to highlight using only simulation methods.

Key words: Formal Verification, Cloud Computing , Theorem Proving , Model Checking

AMS subject classifications. 15A15, 15A09, 15A23

1. Introduction and Backgrounds.

Cloud computing [27] has emerged recently as a new paradigm that moved enterprise computing from the classical host-based architecture pattern into the elastic computing pattern. This new service-oriented vision delivers resources and applications on-demand based on the pay-per-use concept. Cloud computing platforms include two main aspects: hardware infrastructure and software architecture. The first includes communication network equipment, links, data and storage centers, servers, and computational resources. On the other hand, the second includes operating systems, databases, software development platforms, software applications, and middleware. The National Institute of Standards and Technology [58] defined five basic characteristics of cloud computing: on-demand service, fast application elasticity, network access, resource pooling, and measured service. In addition, applications in the cloud are composed of multiple software components running on complex distributed virtual machines, therefore, several management tasks and dedicated protocols are required for configuring and monitoring these distributed applications in order to preserve their consistency.

The cloud system should provide the application users with robustness, fault tolerance, execution automation, and powerful computing facilities. This implies various cloud service requirements to be maintained. Therefore, several verification challenges arise throughout the design development and deployment of these systems. In addition, unlike conventional software and hardware systems, a wide range of different properties and design requirements arise in the cloud based systems. For instance, the consistency of data storage on the cloud, where multiple copies of the data exist at the same time, and the data itself is stored on multiple data centers, is a property that is particular to cloud systems. Additionally, problems arising from incorrect interpretation of the service requirements lead to cloud services that do not conform to the user requirements. Finally, the emergence of new areas such as big data collection and analysis puts more pressure on the need for new verification methods. This explains why the use of formal techniques and tools is necessary in the process of testing and verification of cloud computing systems.

On the other hand, the size of data centers have been continuously increased in order to accommodate the increasing demand while at the same time reducing the management costs. In addition, virtualization has been heavily used in order to increase the utilization of server resources by consolidating many virtual machines into a single physical server. Therefore, cloud computing systems become very complex and dynamic which makes it difficult to manage and test. This lead to various service failures, for instance, business data belonging to

[*]Thanks to Youssef Iraqi, Khalifa University, UAE, for his valuable feedback on the paper.

[†]Department of Electrical and Computer Engineering, Khalifa University, UAE (amjad.gawanmeh@kustar.ac.ae).

[‡]Faculty of Information Sciences & Engineering, École de technologie supérieure Montreal, Québec, Canada (ahmad.alomari.1@ens.etsmtl.ca).

5,700 customers were lost due to the execution of improper operations which occurred at Firstserver Inc. [47]. Amazon Web Services, as another example, halted their services because of invalid configuration changes to their network paths [7]. This incidents happen due to hidden flaws in the cloud system designs that only become visible only after the occurrence of service faults. In order to prevent such situations, thorough testing and verification is required. On the other hand, it has been shown that scheduling algorithms that are being used for cloud frameworks, such as Hadoop for instance, can suffer from scheduling failures due to unforeseen changes in the cloud environments [72]. Conventional simulation methods that are used for this purpose cannot provide full coverage for complex systems, therefore, formal methods are used in addition to simulation to improve the quality and the reliability of cloud systems.

Formal methods [2] use mathematical models for the analysis of computing, communication, and industrial systems in order to establish system correctness with mathematical rigor. Formal methods are highly recommended verification techniques for safety critical systems. Research in formal methods has recently led to the development of promising verification techniques that facilitate the early detection of defects and hence enhance the design quality. These techniques are accompanied by powerful tools that can automate various verification steps which have been successfully used for the analysis of a variety of complex systems [18]. Besides, it has been shown that formal verification would have revealed the exposed defects in, e.g., the Ariane-5 missile, Mars Pathfinder, Intel Pentium II processor, and Therac-25 therapy radiation machine [12]. Existing formal methods include: theorem proving, model checking, and equivalence checking. While the first two have been used recently for the formal verification of cloud systems, the equivalence checking method have only been used in narrow research areas such as digital systems. All formal methods are based on formal syntax and formal semantics, on the other hand, other verification methods adopted formal syntax and allowed informal semantics. These methods are called semi-formal verification methods in the literature.

In this paper, we survey the use of formal methods on testing and verification of cloud systems. This work provides a state of the art reference for reasoning about the correctness of the operation of these systems within their environments. In addition, we identify several open issues in the area and propose directions to handle them. The rest of the paper is organized as follows: Section 2 present theorem proving based methods. In Section 3, we present model checking based methods. Section 4 presents semi-formal methods and other models. In Section 5, we address and discusses open issues. Finally, Section 6 concludes the paper.

2. Theorem proving. Theorem proving [49] is a formal verification method where mathematical logic is used to formulate a theorem about the correctness of a design, then a general purpose theorem-prover is used to construct the proof. The theorem can be expressed using the first-order logic where the proofs can be conducted automatically. Second and higher-order logics are more expressive, and thus can be used to model more complex systems. However, user interaction is required to guide the proof tools. The disadvantage of theorem proving is the need for user interaction to guide the proof tools, and hence, it cannot be fully automated.

The use of theorem proving in modeling and verification of cloud computing systems is very recent. In [42], the authors used Coq [15] formal reasoning system to define and reason about a framework for rewriting Service Level Agreements (SLAs) using set theory to enable efficient co-location in a cloud setting. The authors showed that the framework is consistent with respect to its semantics by providing machine-verified proofs. The work in [6] also used Frama-C software verification tool and Coq theorem prover in order to model and verify the virtual memory system of the cloud hypervisor Anaxagoros that is designed for resource isolation and protection. In [44], and [45], the authors introduced *cloud calculus*, a process algebra based on structural congruence and a reduction relation, for the specification of the migration of virtual machines and security policies in the cloud. The calculus is used to model and verify that the global security policy is preserved after migration in the cloud using Sugar [75].

In [26], the authors presented an abstract formalization of federated cloud workflows using the Z notation [78]. The workflow is modeled as an abstract data type upon which various operations are possible. The Z notation is used to encode the rules for security and cost analysis that enable the calculation of possible workflows using theorem proving and term rewriting. In their work in [76], the authors proposed a cloud-based assured information sharing system in order to define properties such as soundness, transparency, consistency and completeness using Resource Description Framework (RDF). They intend to apply theorem proving using ACL2 [46] to produce machine-checkable proofs that these properties are satisfied by the system. The authors

Table 2.1
Theorem Proving Methods

Ref.	System Model	Verification Tool	Aspect Analyzed	Testbed Arch.	Verified Property
[42]	Set Theory	ARTIFACT-Coq	SLA	None	Safe Transformation
[44]	Cloud calculus	Virtual Machines	Security policy	Amazon EC2	Migrating Virtual Machines
[45]	Cloud calculus	Sugar	Security policy	Amazon EC2	Migrating Virtual Machines
[26]	Z Notation	Z/EVES	Federated Cloud	None	Workflows
[76]	RDF	ACL2	Information Sharing	Hadoop	Access Control
[63]	Event-B	Rodin	Data Consistency	None	Integrity
[6]	ACSL	Frama-C, Coq	Virtual Memory	Anaxagoros	Page Mapping

state no details on how these properties will be modeled or verified.

In a recent work in [63], the authors presented a method based on Event-B for formal modeling of resilient data storage. The work addresses the problem of data consistency in replicated data stores by formally modeling write-ahead logging in replicated data stores in the cloud. Data integrity and consistency properties are considered in three different replication architectures: synchronous, semi-synchronous and asynchronous architectures.

Table 2.1 highlights the main points of the works we surveyed. It describes the formal model, verification tool, cloud testbed architecture, and finally the property or design aspect of the system that was verified. The proposed techniques in theorem proving are still premature and have been only explored for the verification of basic properties for cloud computing systems. In addition there are several techniques and theorems that have been used in the verification of related systems such as distributed and security systems. These theorems could be reused in cloud systems verification if the appropriate cloud model can be devised. Examples of such frameworks are Higher Order Logic (HOL) theorem proving framework [35, 73] and the Prototype Verification System (PVS) [61, 64]. On the other hand, efficient probabilistic theorem proving techniques can be used in formalizing and verifying several properties that are particular to cloud computing such as data consistency and availability, where the focus on one property may affect another.

In fact, the most powerful feature of this technique is reusing proved theorems in constructing proofs of other systems. This issue is not exploited well in the area of cloud computing. Even though theorem proving based techniques require long time and practice, and need expertise in modeling the system and establishing the proof, these deductive verification approaches can handle and verify complex systems due to the expressiveness of this logic. Therefore it can be further explored for modeling and verification of complex properties in cloud systems such as consistency, availability, and trust. For instance, trust can be achieved through reputation establishment that depends on the relation with other nodes in the system, where there are certain thresholds that need to be achieved before trust is granted, while systems have several performance constraints at the same time, therefore, the process of trust establishment can be formalized using probabilistic verification methods in theorem proving, or alternatively, in model checking. Finally, refinement based theorem proving methods, such as Event-B [3] can also be used to model and verify cloud systems at several levels of abstraction using Rodin [25] supporting tool. For instance, big data collection and analysis is an area where new programming and analysis paradigms have emerged. Therefore, new testing and verification techniques need to be developed, yet, there is no work that has addressed this aspect so far. For instance, security in partitioned cloud and security within Amazon EC2 [8] are candidates for theorem proving approach.

The work in also used Coq theorem prover in order to model and verify the virtual memory system of the cloud hypervisor that is designed for resource isolation and protection.

3. Model Checking. Model checking [12] or property checking is a formal verification technique where we check exhaustively and automatically whether a model of a system meets a given specification. Model checking examines all possible system states in a brute-force manner in order to show that a given system model truly satisfies a certain property. The main drawback of model checking approach is the limited size of number of system states that can be checked, therefore, abstraction methods are used to enable the verification of complex and large systems.

Similar to theorem proving methods, the use of model checking in modeling and verification of cloud computing systems is very recent. In addition, most of the methods in the literature used an existing model checker in order to verify a property within the cloud system. In [19], the authors used Petri Nets in order to model a system using three approaches: multithreaded, distributed, and cloud based approach. Then, Time-Basic (TB) nets were used in order to compare their performance in terms of facing the state space explosion problem. In [21], the authors presented a λ calculus model for analyzing and verifying the resources used in web service applications in cloud computing environment. They used Labeled Transition System Analyser (LTSA) [56] to verify the validity of several protection policies. In [1], the authors used SPIN model checker [41] in order to model for Event Condition Action (ECA) in the context of web services and then to verify service agreement property.

In another work in [28], [69], and [68], the authors used parameterized boolean equation systems (PBES) to build a toolbox for verifying distributed processes based on concurrency theory called CADP (Construction and Analysis of Distributed Processes) [29]. The toolbox can handle compiling various formal specification languages, equivalence checking, model checking, compositional verification, and performance evaluation. The tool is illustrated on a self-configuration protocol for distributed applications in the cloud. The work in [77, 74] presented an approach to generate test suites for service choreographies by translating them into Event-B [3] models, and using ProB [53] model checker for the test generation.

In a recent work in [38], a formal framework called vTRUST, is proposed to formally describe virtualization systems with abstraction. The system can be used to verify various properties in virtualization systems such as confidentiality, verifiability, isolation and Platform Configuration Registers (PCR) consistency with respect to certain adversary models. The authors used CSP to model virtualization, and then vTRUST to translate it into the input language of the Process Analysis Toolkit (PAT) model checker [62], which in turn is used to conduct model checking on the properties under test. The method is tested on Trusted Block as a Service (TBaaS) cloud computing implementation. The work in [65] used a similar approach in order to model Hadoop system using CSP and then verify it against deadlock and localization properties using PAT model checker.

On the other hand, the work in [13] used ProVerif [16] model checker to verify the security of cloud based Encrypted Storage Protocols (ESP) against attacks. The authors used the WebSpi web security library to study a series of commercial and academic cryptographic web applications such as SpiderOak, BoxCryptor, and CloudFogger. A recent work in [47] used NuSMV model checker to verify several operational vulnerability properties in a cloud computing system. They first established a formal state based model for cloud systems and its properties, then translated the model into NuSMV syntax, where they conducted model checking to verify a set of properties within a three-tier cloud system that uses Amazon EC2 elastic load balancing between virtual machines.

The work in [57] used High-Level Petri Nets (HLPN) to model and analyze the structural and behavioral properties of three open source VM-based cloud management platforms: Eucalyptus, Open Nebula, and Nimbus. The authors used Satisfiability Modulo Theories Library (SMT-Lib) [14] and Z3 Solver [24] to model about 100 VMs and then verify properties about the consistent creation of VMs. The work in [11] presented TRUSTFOUND, a formal model checking framework for trusted computing platforms that can be used to check platforms on security properties such as confidentiality and attestability, and uncover the implicit assumptions that must be satisfied to guarantee the security properties. The method is based on extending CSP with trust computing and is applied on a cloud computing platform.

Probabilistic model checking has been used in [17] and [48]. In the first, the authors used AVISPA model checker [10] in order to analyze systems security goals in the categories operational correctness, failure resilience, isolation, and security assurance issues such as failure deployment breach. They used a formal language to express high-level security goals based on the Intermediate Format (IF) and set rewriting as formal foundation. The shortage of this approach is ignorance of the probabilistic features of the AVISPA tool and IF language that could be employed in a better way. In the second work, the authors used PRISM probabilistic model checker [51] in order to measure several probabilistic properties such as having more than 4 migration operations retained in a certain sender server. They first conducted experiments in a visualized system to measure the performance characteristics of concurrent Virtual Machines (VM) live migrations. Next, they constructed a model and formally defined quantitative properties formally, then used PRISM in order to determine whether these properties can be satisfied by the constructed performance model. Recently, the work in [59] also used Markov Decision Processes (MDP) in order to model elasticity in cloud computing, and then used PRISM model checker in order to model and verify several elasticity decision policies that aim to maximize user-defined utility functions.

Table 3.1 highlights the main points of the works we surveyed. Model checking suffers from the state-space explosion problem that makes exhaustive verification very difficult for large and complex systems. In addition, it is computationally expensive to cover all the state space of the system model. Therefore, we believe that the use of existing model checking tools limits the verified model of the cloud while trying to fit it into the syntax and semantics of that particular tool. In fact, abstraction is a powerful technique that enables fitting big systems into model checkers, yet, it has not been explored well for modeling and verification of cloud systems. In particular in the area of big data collection and analysis. For instance, CAP theory [34] and big data are potential areas where model checking can be applied.

4. Semi-Formal Methods. While formal verification methods are based on formal syntax and formal semantics, Semi-formal verification methods are based on formal syntax and allow informal semantics. There are several semi-formal languages and supporting tools that have been used for modeling and verification both in academia and industry. For instance, UML and Object Constraints Language (OCL), Petri Nets and its variation, and assertion based verification [4].

In this theme, several approaches have been proposed for testing and verification of cloud computing systems. The work in [71] presents Consistency Verification and Quality Assurance (CVQA) for verifying the consistency of artifacts and data that has been collected through various data collection techniques in SaaS architectures. In [36], the authors used an XML schema in order to model and verify a security policy for Service Oriented Architecture (SOA) protocol in the cloud. In [79], the authors used belief logic to introduce Application-oriented Remote Verification Trust Model (ARVTM) for reasoning about trust and authorization in the cloud. In [50], the authors used Model-based Testing (MBT) to provide a QuickCheck [23] formal description for pre/post conditions and invariant functions for distribute generation of large test data within MapReduce programming model in Hadoop.

In [54], the authors used Interface Automata (IA) to model consistency of service and business processes in SaaS along with transformation rules and algorithm between Business Process Execution Language (BPEL) and the semantic service interface automata model. In [20], the authors used Satisfiability Modulo Theories (SMT) based constraint solving techniques to propose a framework that checks conflicts and inconsistency against policies within user requirements, and selecting appropriate cloud services that satisfy user requirements and comply with the policies in cloud managing systems such as the Monsoon. The work in [66] presented a Trusted MapReduce (TMR) framework based on Trusted Platform Module (TPM) for validating attestation ticket in the cloud.

In [80], the authors proposed a graphical model based on Colored Petri-Nets (CPN) to formally model service contract obligations properties within SaaS and integrate it into Protege [60] including boundedness, liveness, and reversibility. The work in [70] used game theory in order to model retrievability property in cloud systems, however, there is no implementation or tool support. Finally, the work in [37] presented expressions for formal representation and reasoning within a service cloud model. They used coordination language to express SLA interactions and information sharing constraints within cloud context. The work in [52] presented semantic-aware model checking (SAMC) a white-box principle that takes simple semantic information of the target system and incorporates that knowledge into state-space reduction policies where protocols can be specified and validated. The method does not provide any formal semantics hence it is categorized here.

Table 3.1
Model Checking based Methods

Reference	System Model	Verification Tool	Aspect Analyzed	Testbed Arch.	Verified Property
[19]	Petri Nets	TB nets	None	Hadoop	State space explosion
[21]	λ calculus	LTSA	None	Web Service	Resource protections
[1]	Promela	SPIN	ECA	Web Service	Service Agreement
[28]	PBES	CADP	Distributed Proc.	None	Consistency
[69]	LOTOS NT	CADP	Distributed App.	Virtual Machines	Self-configuration
[68] [77]	CSP/Event-B	ProB	None	SOA	Test generation
[13]	CSP	ProVerif	ESP	SaaS	Encrypted Web Storage
[38]	CSP/PAT	PAT	None	TBaaS	PCR Consistency Confidentiality
[65]	CSP/PAT	PAT	None	Hadoop	Deadlock, Locality
[59]	MDP	PRISM	VM	NoSQL	Elasticity
[47]	State-based	NuSMV	Operational vulnerability	EC2	Data Inconsistency, Misconfiguration, faults
[57]	HLPN	SMT-Lib Z3 Solver	VM	Eucalyptus Open Nebula Nimbus	VM consistent creation
[74] [17]	Set Rewriting (IF)	VALID - AVISPA	Virtualized Infrastructure	None	Unreachability Failure Resilience
[11]	TCSP	TrustFound	Security assurance	None	Confidentiality Attestability
[48]	Timed Automata	PRISM	Virtual Machines	None	Migration operations

Table 4.1 highlights the main points of the works we have surveyed. The use of semi-formal methods has shown great success in the aspects that were addressed, however, we believe that there are still several issues that need to be addressed. First, formal methods have been used successfully in the process of test case generation in order to cover certain corner cases in the systems simulation. Hence, this is an area that is still open in cloud computing context. Second, the integration of simulation and formal methods, mainly in assertion based verification, has been a successful method in several applications in the literature. Hence, assertion based verification can be employed in this area. In particular, in the verification of safety properties such as data consistency, and liveness ones such as availability of data in the cloud. Finally, assertion based verification is

Table 4.1
Semi-Formal Methods

Reference	System Model	Verification Tool	Aspect Analyzed	Testbed Arch.	Verified Property
[71]	None	CVQA	none	SaaS	Assurance, consistency
[36]	None	XML	SOAP	Amazon EC2	Security policy
[79]	Belief Logic	ARVTM	TFCC	None	Remote trust
[50]	MBT	VDM	none	Hadoop	Distribution of data
[54]	IA	None	BPEL	SaaS	Consistency
[20]	SMT	zChaff	None	Monsoon	Policy conflict detection
[66]	TPM	TMR	None	Hadoop	Trust: attestation ticket
[80]	Protege	CPN	None	SaaS	Boundedness, liveness, reversibility
[70]	Game theory	None	None	None	Retrievability
[37]	None	None	SLA	None	Information Sharing
[52]	Semantic knowledge	SAMC	None	Hadoop MapReduce	Deep bugs

a well developed technique that has not been explored yet in the verification of cloud systems. For instance, cloud system availability property can be verified using assertion based verification.

5. Discussion and Open Issues. Most of the existing methods addressed the cloud as Software as a Service (SaaS) model, yet other service models have rarely been considered in the verification process, i.e. Infrastructure as a Service (IaaS), Platform as a service (PaaS), or Network as a Service (NaaS). In addition, the use of existing verification tools limits the verified model of the cloud while trying to fit it into the syntax and semantics of that particular tool. Therefore, designing a dedicated framework with proper semantics and syntax that targets cloud based systems is an important issue to be addressed.

Theorem proving methods have not been yet explored well in the context of cloud computing despite the expressiveness of their logic in formalizing systems. In addition, there are several existing techniques that can be reused efficiently in this context. For instance, a lot of work has been conducted in proving security aspects that can be reused here. On the other hand, probabilistic methods have gained quite good attention recently and there are several mature methods in the literature that can assist in developing new techniques for cloud computing based on model checking and theorem proving. In summary, we can identify and discuss four open issues in cloud computing systems where formal methods can be efficiently used.

5.1. Probabilistic Verification of CAP. Consistency, Availability, and Partitions (CAP) [34] theory for cloud computing has been addressed in several works. Formal methods, in particular, probabilistic ones, can be effectively used to verify that the cloud system maintains a certain level of availability or consistency in a partitioned system. For instance, in order to design a partition tolerant cloud system; lack of consistency and/or availability must be tolerated since cloud systems can satisfy the three properties together. In addition, it is required to maintain the consistency and availability tolerance when using data in the cloud. For instance, to prove that a partitioned cloud system can provide a availability while maintaining full consistency is a problem that can be handled in probabilistic model checking. In addition, to prove that the cloud can maintain a minimum level of c consistency and a minimum level of a availability is a more difficult problem that can be handled using probabilistic theorem proving methods such as the one in [39].

The work in [9] presented a method that allows providing linearizable consistency and partition tolerance in cloud systems. It is based on distributing and replicating data according to the principle of consistent hashing to maintain a consistent view of the partitioned cloud system. This work presents an interesting and challenging application where theorem proving technique can be used to formally verify safety and lifeness properties related to the operation of cloud systems including the following properties: partition tolerance, reconfiguration safety, consistent view, termination of reconfiguration, and termination of operations. This shows an interesting cloud related area where theorem proving and/or model checking techniques can be effectively used.

5.2. Assertion based Verification of Safety and Availability. Assertion based verification [4] can be useful in combining simulation and formal methods to verify properties such as safety and availability in the cloud. In fact, multiple instances of the same application can be delivered in a scalable manner. Since failures in the cloud do happen despite the fact that they are rare, application owners are ultimately responsible for availability and recoverability There is a need to balance cost and complexity of high availability efforts against such risks. In fact, increasing availability may affect the security level provided by the cloud system, hence, assertion based verification can provide solution for instances where these security levels are violated. On the other hand, assertions can be used to set a certain level of safety while trying to enhance availability. This helps in detecting safety breaches while trying to achieve a high level of availability.

5.3. Verification of Security in Partitioned Cloud. In order to provide security in the cloud, several developed and well established security methods and protocols were reused. However, this issue may lead into several security problems. For instance, authentication protocols that are designed for typical networks may not be enough in the cloud, since authentication can depend on multiple parts. Another security challenge in cloud systems arise when an application needs to authenticate itself into a cloud data center while the application and the data center are both running on several partitions in the cloud. In order to verify such security properties, a precise formal model for the cloud need to be proposed, and then used to develop verification methods for these properties. In addition, several methods were developed for network security and can be reused in the verification of security properties in the cloud, in particular in theorem proving frameworks such as HOL or PVS.

5.4. Verification of Firewalls in Partitioned Cloud in AWS. Security within Amazon EC2 [8] is provided on multiple levels: the operating system, the virtual, a firewall, and customer security group policies. These security measures are applied in order to prevent data contained within Amazon EC2 from being intercepted by unauthorized systems or users. In addition, the cloud infrastructure should be able to create secure Amazon EC2 instances without sacrificing the flexibility in configuration that customers demand. Firewalls can have policies for groups permitting different classes of instances to have different rules, therefore, this may increase the possibility of animalities between various policies. Since formal methods have been successfully used in the verification of firewalls and security policies, existing methods can be extended to support multiple levels security verification for cloud based infrastructures such as the Amazon EC2, in particular, the correct implementation of customer security group policies. For instance, the work on modeling and verification of firewall rulebase [31, 32], and the work on verification of firewalls with dynamic rulebase update [30, 33].

5.5. Modeling and Verification of Big Data. Big data [67] concept refers to the practice of collection and analysis of huge data sets and the algorithms, tools, and data centers that are used to analyze the massive information associated with this data. Testing and verification of such algorithms is considered costly and challenging due to the huge amount of associated information. Hence, formal methods can provide the necessary background to be able to handle such challenge. In particular, several developed abstraction techniques that have supporting tools in both model checking and theorem proving can be used in this context. These techniques can be efficient in providing abstract models for big data and help in their efficient analysis and verification. For instance, the Event-B method supports abstraction and refinement. Therefore, it can be used to model big data at different levels of abstractions, in particular structured data. In addition, invariant checking in Event-B is suitable for the modeling and verification of several properties related to big data analysis.

On the other hand, the emergence of new programming paradigms such as stream programming, illustrated in the IBM Streams Processing Language (SPL) [40], puts more pressure on the need for reliable abstraction techniques that will lead into efficient testing and verification of several issues related to big data analysis. We believe that verification methods for big data collections and analysis in the cloud could be of great impact because of the lack of any work in this area despite its importance, hence, this issue can have great impact if it is well addressed. Several properties about can be identified and hence need to be modeled and verified under cloud infrastructure. Here we identify the following properties, fault tolerance, scalability, quality, and data realization. Several developed abstraction techniques or stream programming can be used to handle this challenge for their efficiency. Fault tolerance, for example, is a critical issue in the efficiency of big data processing, since non-recoverable data failures may take much time to handle during task scheduling. Hence, model checking can be efficiently used this context, in particular by providing an abstract state based model for big data, and then defining properties about fault tolerance.

Finally, integrity of big data in the cloud has been highlighted as a challenging research problem [55]. Despite that existence of several data integrity verification methods, the cloud owner may still to verify their data that is stored remotely, which is very complex due to the huge amount of data and communication overhead. While proposed solutions suggest to apply mechanisms such as digital signatures on distributed cloud servers to verify data integrity, instead of retrieving the whole data, this methods can be subtle to several critical security problems if the method is not designed properly. In this context, there exists several model checking tools for security protocols that can effectively be used to verify such distributed algorithms. In addition, the processing of large amounts of genomic data requires complex resource and software configuration tasks [22], which adds more challenges to big data manipulation in the cloud. Finally, several emerging issues are currently being integrated into the cloud, for instance, the development frameworks for cloud based medical images processing [43] is a new prominent subject where formal methods can be used in order to enhance the trust in such newly developed systems [5].

6. Conclusions. In this paper we surveyed the topic of using formal methods in testing and verification of cloud systems. Cloud computing systems have emerged recently by moving computing services from the classical host-based architecture pattern into the elastic computing pattern. Therefore, several design and verification challenges have also emerged, such as the consistency of data storage in the cloud. In addition, new research areas have also emerged as a result, such as big data collection and analysis. This justified the need for alternative methods for the verification of these systems since traditional simulation may not be enough. We identified related work in three main categories: theorem proving based methods, model checking based methods, and finally semi-formal methods. We then summarized each category in one table where we showed the underlaying formal model, verification tool, and the cloud related properties. We believe that formal modeling and verification for cloud computing based systems has a positive outlook and involves several challenges, yet, some open issues have not been addressed properly in the literature.

This paper identified possible open issues where formal methods can serve as the underlaying technique in the testing and verification of cloud systems. In the theme of cloud system supporting both consistency and availability, probabilistic theorem proving is very useful in verifying that a cloud system sustains a certain level of assurance about one of these while maintaining the other. On the other hand, assertion based verification techniques can be used for the verification of safety and availability properties. In a new research area, formal models can be efficiently used in big data collections and analysis since abstraction techniques are mature in the literature and can serve well in this area. Finally, new security issues, such as authentication in a partitioned cloud, is an area that needs more investigations.

Formal verification of cloud systems must be integrated into the real time service providing process, and hence, verification techniques need to be developed at the same pace cloud services methods are being advanced. Recent research activities show that the attention is directed towards developing and enhancing the design of cloud systems. However, given the complexity of these systems, and their usage to provide various types of distributed services, their verification is essential during all design, deployment, and service providing stages.

REFERENCES

[1] A. Abdelsadiq, C. Molina-Jimenez, and S. Shrivastava, *A High-Level Model-Checking Tool for Verifying Service Agreements*, in Proc. IEEE Symposium on Service Oriented System Engineering, 2011, pp. 297–304.

[2] J.-R. Abrial, *Faultless Systems: Yes We Can!*, IEEE Computer Journal, 42 (2009), pp. 30–36.

[3] ———, *Modelling in Event-B: System and Software Engineering*, Cambbridge University Press, 2009.

[4] S. Aggarwal, S. Mittal, and S. B. Garg, *Assertion Based Verification*, Soft Computing, (2005), p. 415.

[5] S. Ahmad, O. Hasan, and U. Siddique, *On the formalization of zsyntax with applications in molecular biology*, Scalable Computing: Practice and Experience, 16 (2015).

[6] B. Allan, N. Kosmatov, M. Lemerre, and F. Loulergue, *A case study on formal verification of the Anaxagoros hypervisor paging system with Frama-C*, in International Workshop on Formal Methods for Industrial Critical Systems (FMICS), Oslo, Norway, June 2015, Springer.

[7] Amazon, *Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region, 2011, http://aws.amazon.com/message/65648/*, 2013.

[8] ———, *Amazon Web Services: Overview of Security Processes, 2013, http://aws.amazon.com/security/*, November 2013.

[9] C. Arad, T. M. Shafaat, and S. Haridi, *CATS: Linearizability and Partition Tolerance in Scalable and Self-Organizing Key-Value Stores*, Tech. Report T2012:04 ISSN 1100-3154, Swedish Institute of Computer Science, Sweden, May 2012.

[10] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Drielsma, P. Heam, O. Kouchnarenko, J. Mantovani, S. Madersheim, D. Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Vigana, and L. Vigneron, *The avispa tool for the automated validation of internet security protocols and applications*, in Proc. Computer Aided Verification, vol. 3576 of LNCS, Springer, 2005, pp. 281–285.

[11] G. Bai, J. Hao, J. Wu, Y. Liu, Z. Liang, and A. Martin, *Trustfound: Towards a formal foundation for model checking trusted computing platforms*, in Formal Methods, Springer, 2014, pp. 110–126.

[12] C. Baier and J.-P. Katoen, *Principles of Model Checking*, The MIT Press, Cambridge, MA, USA, 2008.

[13] C. Bansal, K. Bhargavan, A. Delignat-Lavaud, and S. Maffeis, *Keys to the Cloud: Formal Analysis and Concrete Attacks on Encrypted Web Storage*, in Proc. Principles of Security and Trust, D. Basin and J. Mitchell, eds., vol. 7796 of LNCS, Springer Berlin Heidelberg, 2013, pp. 126–146.

[14] C. Barrett, S. Ranise, A. Stump, and C. Tinelli, *The Satisfiability Modulo Theories Library (SMT-LIB) http://www.smtlib.org/, 2013*, 2013.

[15] Y. Bertot and P. Castéran, *Interactive Theorem Proving and Program Development: Coq'Art: the Calculus of Inductive Constructions*, Springer, 2004.

[16] B. Blanchet, M. Abadi, and C. Fournet, *Automated verification of selected equivalences for security protocols*, in Proc. IEEE Symposium on Logic in Computer Science, IEEE, 2005, pp. 331–340.

[17] S. Bleikertz and T. Gross, *A Virtualization Assurance Language for Isolation and Deployment*, in Proc. IEEE Symposium on Policies for Distributed Systems and Networks, 2011, pp. 33–40.

[18] P. Boca, J. Bowen, and J. Siddiqi, *Formal Methods: State of the Art and New Directions*, Springer, 2010.

[19] M. Camilli, *Petri Nets State Space Analysis in the Cloud*, in Proc. IEEE Int. Conference on Software Engineering, 2012, pp. 1638–1640.

[20] C. Chen, S. Yan, G. Zhao, B. S. Lee, and S. Singhal, *A Systematic Framework Enabling Automatic Conflict Detection and Explanation in Cloud Service Selection for Enterprises*, in Proc. IEEE Conference on Cloud Computing, 2012, pp. 883–890.

[21] J. Chen, L. Huang, H. Huang, C. Yu, and C. Li, *A Formal Model for Resource Protections in Web Service Applications*, in Proc. IEEE Cloud and Service Computing, 2012, pp. 111–118.

[22] P. Church and A. Goscinski, *Selected approaches and frameworks to carry out genomic data analysis on the cloud*, Scalable Computing: Practice and Experience, 16 (2015).

[23] K. Claessen and J. Hughes, *QuickCheck: a Lightweight Tool for Random Testing of Haskell Programs*, ACM Notices, 46 (2011), pp. 53–64.

[24] L. De Moura and N. Bjørner, *Z3: An Efficient SMT Solver*, in Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2008, pp. 337–340.

[25] *Rodin Platform, http://www.event-b.org, 2013*, 2013. http://www.event-b.org, 2013.

[26] L. Freitas and P. Watson, *Formalising Workflows Partitioning over Federated Clouds: Multi-level Security and Costs*, in Proc. IEEE World Congress on Services, 2012, pp. 219–226.

[27] B. Furht and A. Escalante, *Handbook of Cloud Computing*, Springer, 2010.

[28] H. Garavel, F. Lang, R. Mateescu, and W. Serwe, *CADP 2011: a Toolbox for the Construction and Analysis of Distributed Processes*, International Journal on Software Tools for Technology Transfer, 15 (2013), pp. 89–107.

[29] H. Garavel, R. Mateescu, F. Lang, and W. Serwe, *Cadp 2006: A toolbox for the construction and analysis of distributed processes*, in Proc. Computer Aided Verification, Springer, 2007, pp. 158–163.

[30] A. Gawanmeh, *Automatic verification of security policies in firewalls with dynamic rule sequence*, in International Conference on Information Technology: New Generations, IEEE Press, 2014, pp. 279–284.

[31] A. Gawanmeh and S. Tahar, *Modeling and Verification of Firewall Configurations Using Domain Restriction Method*, in IEEE International Conference on Internet Technology and Secured Transactions, IEEE Press, 2011, pp. 642–647.

[32] ———, *Novel Algorithm for Detecting Conflicts in Firewall Rules*, in IEEE Canadian Conference on Electrical and Computer Engineering, IEEE Press, 2012, pp. 1–4.

[33] A. Gawanmeh and S. Tahar, *Real time verification of firewalls with dynamic rulebase update*, in IEEE Canadian Conference on Electrical and Computer Engineering, IEEE Press, 2014, pp. 1–6.

[34] S. Gilbert and N. Lynch, *Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services*, ACM SIGACT News, 33 (2002), pp. 51–59.

[35] M. Gordon and T. Melham, *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*, Cambbridge Univ, Press, 1993.

[36] N. Gruschka and L. L. Iacono, *Vulnerable Cloud: SOAP Message Security Validation Revisited*, in IEEE Conference on Web Services, 2009, pp. 625–631.

[37] M. Hale, M. Gamble, and R. Gamble, *A Design and Verification Framework for Service Composition in the Cloud*, in Proc. World Congress on Services, IEEE, June 2013, pp. 317–324.

[38] J. Hao, Y. Liu, W. Cai, G. Bai, and J. Sun, *vTRUST: A Formal Modeling and Verification Framework for Virtualization Systems*, in Proc. Formal Methods and Software Engineering, L. Groves and J. Sun, eds., vol. 8144 of LNCS, Springer, 2013, pp. 329–346.

[39] O. Hasan, S. Tahar, and N. Abbasi, *Formal Reliability Analysis using Theorem Proving*, IEEE Transactions on Computers, 59 (2010), pp. 579–592.

[40] M. Hirzel, H. Andrade, B. Gedik, G. Jacques-Silva, R. Khandekar, V. Kumar, M. Mendell, H. Nasgaard, S. Schneider, R. Soule, et al., *IBM Streams Processing Language: Analyzing Big Data in Motion*, IBM Journal of Research and Development, 57 (2013), pp. 7–1.

[41] G. J. Holzmann, *The Model Checker SPIN*, IEEE Transactions on Software Engineering, 23 (1997), pp. 279–295.

[42] V. Ishakian, A. Lapets, A. Bestavros, and A. Kfoury, *Formal Verification of SLA Transformations*, in Proc. IEEE World Congress on Services, 2011, pp. 540–547.

[43] C. Jansen, M. Beier, M. Witt, J. Wu, and D. Krefting, *Extending xnat towards a cloud-based quality assessment platform for retinal optical coherence tomographies*, Scalable Computing: Practice and Experience, 16 (2015).

[44] Y. Jarraya, A. Eghtesadi, M. Debbabi, Y. Zhang, and M. Pourzandi, *Cloud Calculus: Security Verification in Elastic Cloud Computing Platform*, in Proc. IEEE Collaboration Technologies and Systems, 2012, pp. 447–454.

[45] ———, *Formal verification of security preservation for migrating virtual machines in the cloud*, in Proc. Stabilization, Safety, and Security of Distributed Systems, A. Richa and C. Scheideler, eds., vol. 7596 of LNCS, Springer, 2012, pp. 111–125.

[46] M. Kaufmann and S. Moore, *ACL2: an Industrial Strength Version of Nqthm*, in Proc. IEEE Computer Assurance, Systems Integrity, Software Safety, Process Security, 1996, pp. 23–34.

[47] S. Kikuchi and T. Aoki, *Evaluation of Operational Vulnerability in Cloud Service Management Using Model Checking*, in International Symposium on Service Oriented System Engineering, IEEE, March 2013, pp. 37–48.

[48] S. Kikuchi and Y. Matsumoto, *Performance modeling of concurrent live migration operations in cloud computing systems using prism probabilistic model checker*, in Proc. IEEE International Conference on Cloud Computing, 2011, pp. 49–56.

[49] T. Kropf, *Introduction to Formal Hardware Verification*, Springer, 1999.

[50] S. Kusakabe, *Large Volume Testing for Executable Formal Specification Using Hadoop*, in Proc. Parallel and Distributed Processing Workshops and Phd Forum, 2011, pp. 1250–1257.

[51] M. Kwiatkowska, G. Norman, and D. Parker, *PRISM: Probabilistic symbolic model checker*, in Proc. Computer Performance Evaluation: Modelling Techniques and Tools, vol. 2324, Springer, 2002, pp. 200–204.

[52] T. Leesatapornwongsa, M. Hao, P. Joshi, J. F. Lukman, and H. S. Gunawi, *Samc: Semantic-aware model checking for fast discovery of deep bugs in cloud systems*, in USENIX Symposium on Operating Systems Design and Implementation, USENIX Association, 2014, pp. 399–414.

[53] M. Leuschel and M. Butler, *ProB: a Model Checker for B*, in Formal Methods Europe, Springer, 2003, pp. 855–874.

[54] J. Li, S. Chen, L. Jian, and H. Zhang, *A Web Services Composition Model and Its Verification Algorithm Based on Interface Automata*, in Proc. IEEE Conference on Trust, Security and Privacy in Computing and Communications, 2011, pp. 1556–1563.

[55] C. Liu, R. Ranjan, X. Zhang, C. Yang, and J. Chen, *A big picture of integrity verification of big data in cloud computing*, in Handbook on Data Centers, Springer, 2015, pp. 631–645.

[56] J. Magee, J. Kramer, R. Chatley, S. Uchitel, and H. Foster, *LTSA–Labelled Transition System Analyser.* http://www.doc.ic.ac.uk/ltsa/, 2012.

[57] S. Malik, S. Khan, and S. Srinivasan, *Modeling and Analysis of State-of-the-art VM-based Cloud Management Platforms*, Cloud Computing, IEEE Transactions on, 1 (2013), pp. 1–1.

[58] P. Mell and T. Grance, *The NIST Definition of Cloud Computing.* National Institute of Standards and Technology, USA, September 2011.

[59] A. Naskos, E. Stachtiari, A. Gounaris, P. Katsaros, D. Tsoumakos, I. Konstantinou, and S. Sioutas, *Cloud elasticity using probabilistic model checking*, 2014.

[60] N. F. Noy, M. Crubézy, R. W. Fergerson, H. Knublauch, S. W. Tu, J. Vendetti, and M. A. Musen, *Protege-2000: an open-source ontology-development and knowledge-acquisition environment*, in Proc. AMIA Annual Symposiom, 2003, p. 953.

[61] S. Owre, J. M. Rushby, and N. Shankar, *PVS: A Prototype Verification System*, in Proc. Automated Deduction, Springer, 1992, pp. 748–752.

[62] PAT, Process Analysis Toolkit, *http://www.comp.nus.edu.sg/, 2013*, 2013. http://www.comp.nus.edu.sg/, 2013.

[63] I. Pereverzeva, L. Laibinis, E. Troubitsyna, M. Holmberg, and M. Pöri, *Formal Modelling of Resilient Data Storage in Cloud*, in Formal Methods and Software Engineering, L. Groves and J. Sun, eds., vol. 8144 of LNCS, Springer, 2013, pp. 363–379.

[64] PVS Specification and Verification System, *http://pvs.csl.sri.com/, 2013*, 2013. http://pvs.csl.sri.com/, 2013.

[65] G. Reddy, Y. Feng, Y. Liu, J. S. Dong, S. Jun, and R. Kanagasabai, *Towards Formal Modeling and Verification of Cloud Architectures: A Case Study on Hadoop*, in Proc. IEEE World Congress on Services, IEEE, 2013, pp. 306–311.

[66] A. Ruan and A. Martin, *TMR: Towards a Trusted MapReduce Infrastructure*, in Proc. IEEE World Congress on Services, 2012, pp. 141–148.

[67] P. Russom, *Big data analytics*, TDWI Best Practices Report, Fourth Quarter, (2011).

[68] G. Salaün, F. Boyer, T. Coupaye, N. De Palma, X. Etchevers, and O. Gruber, *An Experience Report on the Verification of Autonomic Protocols in the Cloud*, Innovations in Systems and Software Engineering, (2013), pp. 1–13.

[69] G. Salaün, X. Etchevers, N. De Palma, F. Boyer, and T. Coupaye, *Verification of a Self-configuration Protocol for Distributed Applications in the Cloud*, in Proc. of the ACM Symposium on Applied Computing, ACM, 2012, pp. 1278–1283.

[70] H. Shacham and B. Waters, *Compact Proofs of Retrievability*, Journal of Cryptology, (2012), pp. 1–42.

[71] S. Singh and I. Chana, *Consistency Verification and Quality Assurance (CVQA) Traceability Framework for SaaS*, in Proc. IEEE International Advance Computing Conference, 2013, pp. 1–6.

[72] M. Soualhia, F. Khomh, and S. Tahar, *Predicting Scheduling Failures in the Cloud*, Tech. Report Concordia-07-13-2015, Concordia University, Montrea, Canada, July 2015.

[73] H. Sourceforge Project. The HOL System Reference, *http://hol.sourceforge.net, 2013*.

[74] A. Stefanescu, S. Wieczorek, and M. Schur, *Message Choreography Modeling - A Domain-Specific Language for Consistent Enterprise Service Integration*, Software & Systems Modeling, (2012), pp. 1–25.

[75] N. Tamura and M. Banbara, *Sugar: A CSP to SAT translator based on order encoding*, in Proceedings of the Second International CSP Solver Competition, 2008, pp. 65–69.

[76] B. Thuraisingham, V. Khadilkar, J. Rachapalli, T. Cadenhead, M. Kantarcioglu, K. Hamlen, L. Khan, and F. Husain, *Towards the Design and Implementation of a Cloud-centric Assured Information Sharing System*, Tech. Report UTDCS-27-11, The University of Texas at Dallas, September 2011.

[77] S. Wieczorek, V. Kozyura, A. Roth, M. Leuschel, J. Bendisposto, D. Plagge, and I. Schieferdecker, *Applying Model Checking to Generate Model-based Integration Tests from Choreography Models*, in Proc. International Conference on Testing of Communicating Systems, vol. 5826, Springer, 2009, pp. 179–194.

[78] J. Woodcock and J. Davies, *Using Z: Specification, Refinement, and Proof*, Prentice-Hall, Inc., 1996.

[79] X. Zhang, H. Liu, B. Li, X. Wang, H. Chen, and S. Wu, *Application-Oriented Remote Verification Trust Model in Cloud Computing*, in Proc. IEEE Conference on Cloud Computing Technology and Science, 2010, pp. 405–408.

[80] J. Zou, Y. Wang, and K.-J. Lin, *A Formal Service Contract Model for Accountable SaaS and Cloud Services*, in IEEE International Conference on Services Computing, 2010, pp. 73–80.

Edited by: Dana Petcu
Received: May 31, 2015
Accepted: Aug 23, 2015

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:
- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:
- programming environments,
- debugging tools,
- software libraries.

Performance:
- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:
- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:
- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in $\LaTeX\,2_\varepsilon$ using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.