Increasing the Robustness of Self-Organizing Systems By Means of Trust Practices

Dissertation

for the degree of
Doctor of Natural Sciences (Dr. rer. nat.)

submitted to the
Department of Computer Science

University of Augsburg

presented by

Nizar Msadek

in 2016

Thesis entitled: Increasing the Robustness of Self-Organizing Systems By Means of Trust Practices.

Supervisor: **Prof. Dr. Theo Ungerer**, Department of Computer Science
University of Augsburg, Germany

Adviser: **Prof. Dr. Jörg Hähner**, Department of Computer Science
University of Augsburg, Germany

Examiner: **Prof. Dr. Robert Lorenz**, Department of Computer Science
University of Augsburg, Germany

Date of defense: 23-02-2017

Augsburg, October 2016

Abstract

Self-organizing systems are becoming increasingly complex in their organisational structures, especially when unknown heterogeneous entities might arbitrarily enter and leave the network at any time. Therefore, new ways have to be found to develop and manage them. One way to overcome this issue is trust. Using appropriate trust mechanisms, entities in the system can have an indication about which entities to cooperate with. This is very important to improve the robustness of self-organizing systems, which depends on a cooperation of autonomous entities. The contributions of this thesis are trustworthy concepts and generic self-* properties that work in a distributed manner and with no central control to ensure robustness. The self-* properties examined in this thesis are self-configuration, self-optimization, and self-healing. We believe that these properties are fundamental for the design of every autonomous, scalable and fault-tolerant self-* middleware. The self-configuration is related to the ability to perform an initial distribution of services on nodes taking the resource requirement and importance level of services into account. The self-optimization focuses on optimizing the allocation of services at runtime by monitoring the trust and resource consumption of nodes. And the self-healing aspect is concerned with the ability to handle unexpected disturbances in the system in order to guarantee that all services running on nodes stay available even in case of crash, execution and reachability failures.

The resulting middleware is TEM, a trust-enabling middleware that can profit from the advantage of trust and self-* properties at the same time. The TEM makes use of different trust metrics, i.e., such as direct trust, reputation, and confidence to monitor the behavior of nodes in the system. These techniques have been used to improve the robustness of the self-* properties, both at design time and at runtime. Moreover, we applied the TEM middleware to different application case studies and clarified how uncertainty in open environments can be mastered by using the approaches investigated in this thesis. Due to the fact that future application services will become more autonomous, we expect to see more self-organizing systems based on our (or similar) approaches. The investigations conducted within this thesis are a step in this direction.

Acknowledgements

This thesis would not have been possible without the support of many people:

First and foremost, I would like to express my sincere gratitude to my supervisor **Prof. Dr. Theo Ungerer**, director of the System and Networking department. He has been supportive since the days I began working on the OC-Trust project. I would like to thank him for encouraging my research and for allowing me to grow as a research scientist. His advices on both research as well as on my career have been priceless.

I would also like to thank my adviser **Prof. Dr. Jörg Hähner** for his valuable scientific discussions we had during the OC-Trust project and **Prof. Dr. Robert Lorenz** for accepting to be an examiner for my thesis.

My sincere thanks go to **all my colleagues** at the University of Augsburg, for their many interesting discussions, supports and comments on my work, especially **Dr. Rolf Kiefhaber** for his technical advices and his help in reviewing the publications.

Special thanks go to all members of the **OC-Trust research community** for inspiring me with their excellent research.

I am grateful for my sources of funding: the priority program 1085 "OC-Trust" of the **DFG** and the **Bavarian state government**.

Last but not least I would like to thank my parents **(Kamel** & **Rafika**) for their education and support during my studies, my sisters (**Ines** & **Syrine**) and my **friends** for their continuous help. I cannot forget my wife **Nadine** who has encouraged me to do my best in all matters of life.

Nizar Msadek,
Augsburg, October 2016

Dedicated to my family, Bismillah

Contents

1 Introduction

1.1 Motivation

The growing complexity of today's computing systems requires a large amount of administration, which poses a challenging task for manual administration. Initiatives such as Autonomic Computing [JDM03] and Organic Computing [MS04] aim to control these complexities by introducing so-called self-* properties like self-configuration, self-optimization, self-healing and self-protection. These properties can be achieved by constantly observing the system and initiating autonomous reconfigurations when necessary, as implemented by the MAPE cycle [HAM08] and the Observer/Controller paradigm [UMJ+06]. An essential aspect that becomes particularly prominent in this kind of systems is trust [RAS+16]. As part of this work, a trustworthy design of self-* properties for Autonomic and Organic computing systems is investigated. By utilizing trust as a constraint in such systems, the self-* properties would be able to consider the behavior of participant nodes and to maintain a more robust configuration in the face of unreliable components. This enables building a reliable systems out of unreliable components.

1.2 Research Context and Origin

Trust is an important aspect in human societies. It enables cooperation and provides means to estimate potential cooperation partners. Several works were done

to transfer the concept of trust to computer systems. This dissertation is part of the research unit OC-Trust [1] of the German Research Foundation (DFG). We introduced our definition of trust in [SKL+10]. We see trust as a multifaceted multicontextual subject [KJMU13]. Therefore, prior to this dissertation, trust metrics according to [SKL+10] have been developed by Rolf Kiefhaber in the course of his Phd thesis [Kie14]. These metrics are integrated into the TEM [ASM+13], which is a trust enabling middleware developed by the OC-Trust team. The TEM is programmed in Java and provides all tools and concepts to build the trust-enhanced self-* properties. The main trust metrics considered in this work are *direct trust* and *reputation*, which are presented as following.

- **Direct Trust [KSS+10]** is based on the own experiences a node has made directly with an interaction partner node. Typically, trust values are calculated by taking the mean or weighted mean of past experiences.
- **Reputation [KHS+11]** is based on the trust values of others that had experiences with the interaction partner. Reputation is typically collected if not enough or outdated own experiences exist.

When all the aforementioned values are obtained, a total trust value based on direct trust and reputation values can be calculated using confidence [KAS+12] to weight both parts against each other [KJMU13]. Figure 1.1 depicts the relation of the different trust aspects, i.e., direct trust, reputation, and confidence to get a total trust value. This value is then used to enhance self-* properties with trust capabilities. In my thesis, I primarily focus on developing trust-aware self-* properties that work in a distributed manner and also ensure global optimality.

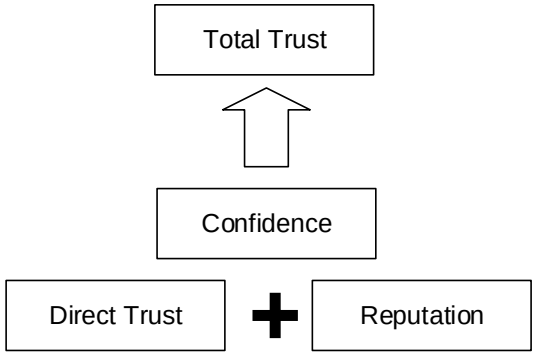


Figure 1.1: Aggregating direct trust with confidence and reputation to a total trust value

[1] OC-Trust: is an acronym of a research unit sponsored by the DFG that deals with the trustworthiness of Organic Computing systems.

1.3 Application scenario and contributions

The self-organizing systems considered in this work are grid systems [FKT01] with nodes representing client machines, which can interact with each other through a set of messages. The system is distributed without a global control. The considered applications are composed of services, which are distributed among the network. Such a scenario would be suited for systems where clients run applications that produce large amount of services and thus are in high demand of computing resources, such as modern applications of Internet of Things [AIM10] or Industry 4.0 [LFK+14]. The services are categorized into important services and non important services. Important services are those, which are necessary for the functionality of the entire system. However, unimportant services are those which have only a low negative effect on the entire system if they fail. The baseline self-* properties introduced in [TKU06, SRK+11, SU08, ASTU06] are proven to be good solutions to reduce the complexity of such systems but these are based on a benevolence assumption that all parts of the system are trustworthy and interested to further the system goal. In open and heterogeneous systems where arbitrary node participants can join the systems, this benevolence assumption must be dropped, since such a participant node may act maliciously and try to exploit the system. This introduces a not yet considered uncertainty, which needs to be addressed using the trust metrics presented in Section 1.2.

1.4 Research challenges

This thesis is concerned with the identification of research challenges related to the construction of self-* properties capable of acting trustworthy in open and distributed environments, as well as to propose methods of resolution for them.

1.4.1 Trust-Aware Self-Configuration

Future computing systems are comprised of a large amount of devices —or nodes for short— so that it would be impossible for a manual system administration to configure them individually. The solution for this issue is to make such systems self-configuring. This means the ability (1) to perform an initial distribution of services on nodes and (2) to perform a reconfiguration of running services due to self-optimization or self-healing demands, i.e., see below, the dependency of the self-* properties.

There are many sophisticated approaches to deal with the initial distribution of services on nodes, either to achieve good load balancing or to minimize energy con-

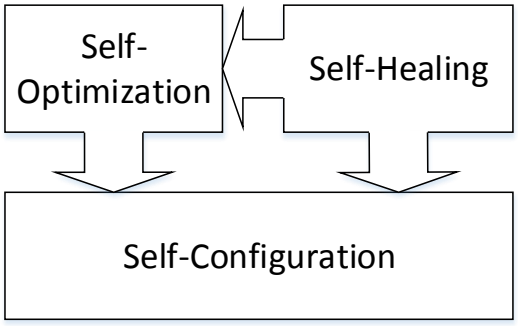


Figure 1.2: Dependency of self-* properties

sumption. An approach that has become a standard by FIPA [2] is the Contract Net Protocol [Smi80]. It consists of finding an agent that is the most suitable to provide a service. This approach is often adapted and applied in many application domains, for example, manufacturing systems [HC09], resource allocation in grids and sensor web environments [KB09, GG08], as well as in hospitals [DGB03], electronic marketplaces [DKRA00] or power distribution network restoration [KHS+09]. Our approach of Chapter 4.3 is based on the Contract Net Protocol, extended by trust. In this context, trust serves as a mean to give nodes an indication about with which nodes to cooperate. Considering the trust constraint of nodes, our approach aims on the one hand to equally distribute the load of services on nodes as in a typical load balancing scenario and on the other hand to assign services with different importance levels to nodes so that the more important services are assigned to more trustworthy nodes. Evaluation results based on a TEM simulation showed that the proposed approach is able to improve the availability of important services. However, self-configuration must take into account the termination time needed for assigning services not only by using one manager but also with multiple managers. Therefore, we suggested in Chapter 4.5 a way to extend the self-configuration with multiple managers to achieve a simultaneous behavior of assigning services on nodes.

1.4.2 Trust-Aware Self-Optimization

Self-organizing systems should be able to dynamically adapt their behavior in response to changes in their environment. At runtime, they should have the ability to deal with situations not anticipated in design time, since not every situation has to be considered when designing the system. After the initial service distri-

[2] FIPA: Interaction Protocol Specifications [Accessed: March 18, 2015] `http://www.fipa.org/specs/fipa00029/`

bution that is given using the self-configuration process, nodes must be able to constantly observe their current resource consumptions as well as the trustworthiness of nodes they are cooperating with, identify unacceptable situations and reconfigure themselves to regain an acceptable state. Therefore, in Chapter 5 a self-optimization algorithm is presented to optimize during runtime the allocation of services on nodes. The algorithm does not only consider pure load-balancing but also takes into account trust to improve the assignment of important services to trustworthy nodes. More precisely, it uses different optimization strategies to determine whether a service should be transferred to another node or not. The self-optimization process is investigated based on the TEM simulation. The evaluation results showed that the found optimization solutions regarding trust and workload are best efforts, in the sense that they make a trade-off problem in which it is impossible to make any trust distribution better without making at least the load balancing distribution worse. Moreover, the evaluation results showed that the proposed self-optimization works only well when nodes in the system have the same trust value in a certain node, i.e., without conflicting trust values. However in a real life situation, this assumption must be dropped, since nodes are free to have the same trust opinion about other nodes or not. Therefore, in Chapter 6 a conflict resolution mechanism is proposed as an extension to the self-optimization process to operate with conflicting trust values at the same time. Moreover, other investigations are done in Chapter 5.6 to accelerate the processing time of the self-optimization within the TEM simulation.

1.4.3 Trust-Aware Self-Healing

Self-healing can be defined as the property that enables a system to perceive services that are not operating correctly and, without human intervention, make the necessary adjustments to restore them using the aforementioned self-configuration and self-optimization principles. Two ways of thinking have to be considered in the self-healing process, which are (1) the proactive self-healing and (2) the reactive self-healing.

- **Regarding (1):** Enables to detect node instability prior to fail. This is generally recognized by observing the degradation of a node trust value. Then, the system moves all running services by using self-optimization techniques to a more reliable node. For this purpose, a number of strategies are proposed in Chapter 5 to allow an autonomous organization of the proactive self-healing and to avoid false proactive shifts during runtime.
- **Regarding (2):** Nodes save recovery information periodically during failure

free execution. Upon failure, which has to be detected by using a failure detector, a node uses the already saved information to restart from an intermediate state i.e., called Snapshot, thus reducing the amount of lost computation. For this purpose, a generic failure detector approach base on trust is proposed in Chapter 7. It aims to improve the failure detection delays by adapting to the trust conditions of the network. Moreover, the proposed approach aims to reduce the number of messages that arises from sending heartbeat messages. An evaluation is provided for the monitoring approach and the results attest a very good detection quality in the network compared to other approaches. Another investigation that is mainly concerned with point (2) is the management of replicas. Chapter 8 details the developed approach used for this topic. It aims to calculate the minimum number of replicas needed for a desired degree of availability taking into account the average availability of nodes. Also here, an evaluation is provided with respect to TEM and the results show a better trust distribution for important replicas with a significant reduction in overhead when compared to the baseline.

1.5 Outline

The concrete contributions of this thesis are trust practices that aim to improve the robustness of self-organizing systems in open environments and a middleware with self-* properties addressing the aforementioned challenges. The self-* properties examined in this work are self-configuration, self-optimization, and self-healing, because we believe that these properties are fundamental for the design of every autonomous, scalable and fault-tolerant middleware. The main investigations on these concepts are given in chapters 2 to 8, and every chapter is arranged in a way that is self-explanatory and separately readable. However, for better understanding the thesis, it is recommended to follow the reading sequence illustrated in Figure 1.3. Chapter 2 deals with the necessary basic knowledge to understand the research area of self-organizing systems, especially self-* systems belonging to the category of Organic and Autonomic Computing Systems. Readers that feel already familiar with these systems, may skip chapter 2 or at least some parts of it. Chapter 3 introduces the baseline system considered in this work. It discusses also the architectural design of our TEM middleware that we have developed to host the trustworthy self-* layer in order to make it more robust in face of untrustworthy components. The trust-aware self-configuration is addressed in Chapter 4. It is related to the ability to perform an initial distribution of services on nodes taking the resource requirement and importance level of services into account. Then, Chapter 5 presents the trust-aware self-optimization, which focuses on optimizing

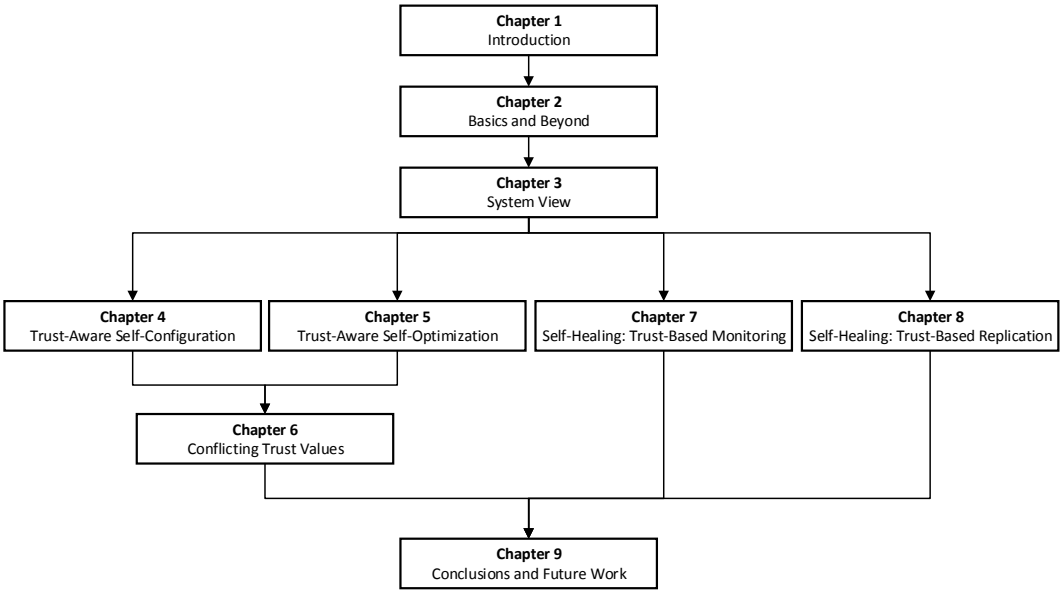


Figure 1.3: Structure of the Thesis.

the allocation of services at runtime by monitoring the trust and resource consumption of nodes. Chapter 6 provides a conflict resolution mechanism for the case that conflicting trust values occur at runtime. The self-healing concepts are given later in Chapters 7 and 8 to deal with the problem of failure monitoring and replication management, respectively. Finally, Chapter 9 summarizes this thesis and gives an outlook on future work.

1.6 Published Materials for the Purpose of this Thesis

Parts of the contents of this thesis have been published by the author in several journals, conferences, workshops, and book chapters. The most important publications are listed below:

Journals

- **[MU17b]:** Nizar Msadek and Theo Ungerer. *Trustworthy Self-Optimization for Organic Computing Environments Using Multiple Simultaneous Requests*. In JSA 2017: Journal of Systems Architecture, issn 1383-7621, `http://www.sciencedirect.com/science/article/pii/S1383762117301388`
- **[MKU15a]:** Nizar Msadek, Rolf Kiefhaber, and Theo Ungerer *A Trustworthy, Fault-tolerant and Scalable Self-Configuration Algorithm for Organic Computing*

Systems. In JSA 2015: Journal of Systems Architecture, issn 1383-761, `http://www.sciencedirect.com/science/article/pii/S138376211500082X`

Conferences

- **[MKU15b]:** Nizar Msadek, Rolf Kiefhaber, and Theo Ungerer. *Trustworthy Self-Optimization in Organic Computing Environments*. In ARCS 2015: Proceedings of the 28th International Conference on Architecture of Computing Systems, pages 123-134, Porto, Portugal, Springer 2015.
- **[MKFU14]:** Nizar Msadek, Rolf Kiefhaber, Bernhard Fechner, and Theo Ungerer. *Trust-Enhanced Self-Configuration for Organic Computing Systems*. In ARCS 2014: Proceedings of the 27th International Conference on Architecture of Computing Systems, pages 37-48, Lübeck, Germany, Springer, 2014.
- **[MKU14b]:** Nizar Msadek, Rolf Kiefhaber, and Theo Ungerer. *A Trust- and Load-Based Self-Optimization Algorithm for Organic Computing Systems*. SASO 2014: Proceedings of the 8th International Conference on Self-Adaptive and Self-Organizing Systems, pages 177-178, London, England, IEEE Computer Society, 2014.

Workshops

- **[MU16b]:** Nizar Msadek and Theo Ungerer. *Trust-Based Monitoring for Self-Healing of Distributed Real-Time Systems*. The 7th IEEE Workshop on Self-Organizing Real-Time Systems (SORT16) in conjunction with ISORC 2016, pages 177-178, York, England, IEEE Computer Society, 2016.
- **[MSKU15]:** Nizar Msadek, Alex Stegmeier, Rolf Kiefhaber, and Theo Ungerer *A Mechanism for Minimizing Trust Conflicts in Organic Computing systems*. In SAOS 2015: Proceedings of the second International Workshop on Self-Optimisation in Organic and Autonomic Computing Systems in conjunction with ARCS 2015, pages 1-7, Porto, Portugal, IEEE Computer Society, 2015.
- **[MKU14a]:** Nizar Msadek, Rolf Kiefhaber, and Theo Ungerer. *Simultaneous Self-Configuration with Multiple Managers for Organic Computing Systems*. In SAOS 2014: Proceedings of the second International Workshop on Self-Optimisation in Organic and Autonomic Computing Systems in conjunction with ARCS 2014, pages 1-7, Lübeck, Germany, IEEE Computer Society, 2014.

Book Chapters

- **[MU16a]:** Nizar Msadek and Theo Ungerer. *Trust as Important Factor for Building Robust Self-x Systems*. Book chapter in Trustworthy Open Self-Organising Systems, Autonomic Systems series, Vol. 7, 2016, Springer International Publishing, pp 153-183, `http://www.springer.com/de/book/9783319291994` ISBN: 978-3-319-29199-4

- **[Msa15]:** Nizar Msadek. *Trust as a principal ingredient to improve the robustness of self-organizing systems*. Book chapter in Organic Computing, Doctoral Dissertation Colloquium, Vol. 7, 2016, Kassel University Press, pp 35-42, `http://www.upress.uni-kassel.de/katalog/abstract.php?978-3-7376-0028-6` ISBN: 978-3-7376-0028-6

Papers Under Review

- **[MU17a]:** Nizar Msadek and Theo Ungerer. *An Efficient Replication Approach Based on Trust for Distributed Self-Healing Systems*. In ICINCO 2017: Proceedings of the of the 14th International Conference on Informatics in Control, Automation and Robotics, (Submitted but not yet published), Madrid, Spain, Springer 2017.

2

Basics and Beyond

Abstract. This chapter provides the necessary background knowledge related to the field of self-organizing systems in general and Autonomic and Organic Computing Systems in particular. This is very important to understand terms, definitions and approaches mentioned and investigated in this thesis. The chapter clarifies also the difference between self-* properties such as self-configuration, self-optimization and self-healing and discusses their limitation to cope with open environments. Furthermore, it introduces the social concept of trust as a solution for that limitation and explains how it can be used in such systems to achieve more robustness. Finally, the chapter surveys other research fields related to the thesis.

2.1 Introduction

The growing complexity of computer systems as well as the high need for self-organization has led scientists to focus more on self-organizing systems. These systems address self-organization in various concerns including robustness, resilience, stability, scalability, performance etc. While self-organizing systems are used in a very large number of different areas, this chapter focuses only on their application in open distributed environments, called open distributed self-organizing systems. It is within this context that several new research keys have emerged over the

years, for instance self-* systems including both of Organic and Autonomic Computing systems [MS04, Hor01], Trusted Computing Systems [Mül08], Self-aware Computing Systems [LPR+16], and System of Systems [Jam08]. Even though every research key has its own field, many of them are often combined with other fields to provide more robust and efficient solution for managing complexity. This can be inferred from the workshop and conference series that were held at the topic of self-* system during the last 10 years: for example the International Conference on Autonomic Computing (ICAC), the International Conference on Architecture of Computing Systems (ARCS), the International Conference on Self-Organizing and Self-Adaptive Systems (SASO), the International Conference on Autonomic and Trusted Computing (ATC), the International Workshop on Self-optimization in Autonomic and Organic Computing Systems (SAOS), or the International Workshop on Self-Improving System Integration (SISSY), to name a few.

In this chapter we give a survey of much that is known about self-* systems. The first Section 2.2 introduces the necessary terms and definitions related to self-* systems along with Autonomic and Organic Computing. Moreover, it focuses on techniques known in the literature as basis to clarify the approaches mentioned and developed in this thesis. In Section 2.4, we survey the role of trust in our work and how it can be used for achieving more robustness and efficiency in open distributed self-* systems.

2.2 Autonomous Self-* Systems

Understanding the different types of self-* systems depends strongly on the research area in which they appear. In this section, we focus on the modern uses of Autonomic and Organic Computing systems as examples to talk reasonably about distributed self-* systems. Another related research areas are then introduced at the end of this section.

2.2.1 Autonomic Computing

Autonomic Computing (AC) [Hor01] is an initiative proposed by IBM since 2001, whose main inspiration comes from the human nervous autonomic system. In this sense, an autonomic system would control the processing of computer applications without intervention of the user as in the same manner that the autonomic nervous system controls vital low level functions (e.g., respiration, heart rate, and blood pressure) without conscious input from the individual. This is achieved by using high level policies which are given either at design time or changed at runtime.

The goal of AC is more focused on data centers in order to allow server systems to be self-managing, i. e., self-configuring, self-optimizing, self-healing, and self-protecting.

- Achieving a self-configuring goal such as installing software when it recognizes that some prerequisite software is missing
- Achieving a self-optimizing goal such as regulating the current workload when it observes an increase or decrease in capacity
- Achieving a self-healing goal such as correcting a configured path so installed software can be correctly located
- Achieving a self-protecting goal such as taking resources offline if it detects an intrusion attempt

These self-* properties [1] can also be orchestrated with each other allowing the system to provide a high degree of autonomy while keeping the system complexity invisible to the user. Although initially equipped only with these four self-* properties, additional properties were latter extended to AC such as self-awareness, self-situation, self-monitoring, and self-adjustment [DSNH10]. To support the effective development of AC systems, a number of design architectures have been proposed over the years. Figure 2.1 depicts one of the most well-known reference architecture proposed by IBM [IBM06]. This is composed of four building blocks which are briefly in the following explained:

The lowest building block incorporates the different managed resources (e.g., servers, storage, applications, etc.) that make up the complete IT infrastructure of a company or an organization. It does not matter if these managed resources have already embedded self-managing properties or not. The next block contains consistent, standard manageability interfaces for accessing and controlling the managed resources. These interfaces are delivered by so-called touchpoints. Layers three and four automate some portion of the IT process using an autonomic manager that can implement a control loop to enable self-configuration, self-healing, self-optimization, and self-protection. Finally, the top building block is responsible for manual administration, typically represented by a human activity such as an administrator to manage and collaborate the autonomic managers with each other if it is needed.

In order to enable this, every autonomic manager has to implement an intelligent control loop — known as the Monitor-Analyze-Plan-Execute control loop or

[1] Sometimes also referred to as self-* principles or self-x properties (self-configuring, self-healing, self-optimizing, self-healing and self-protection)

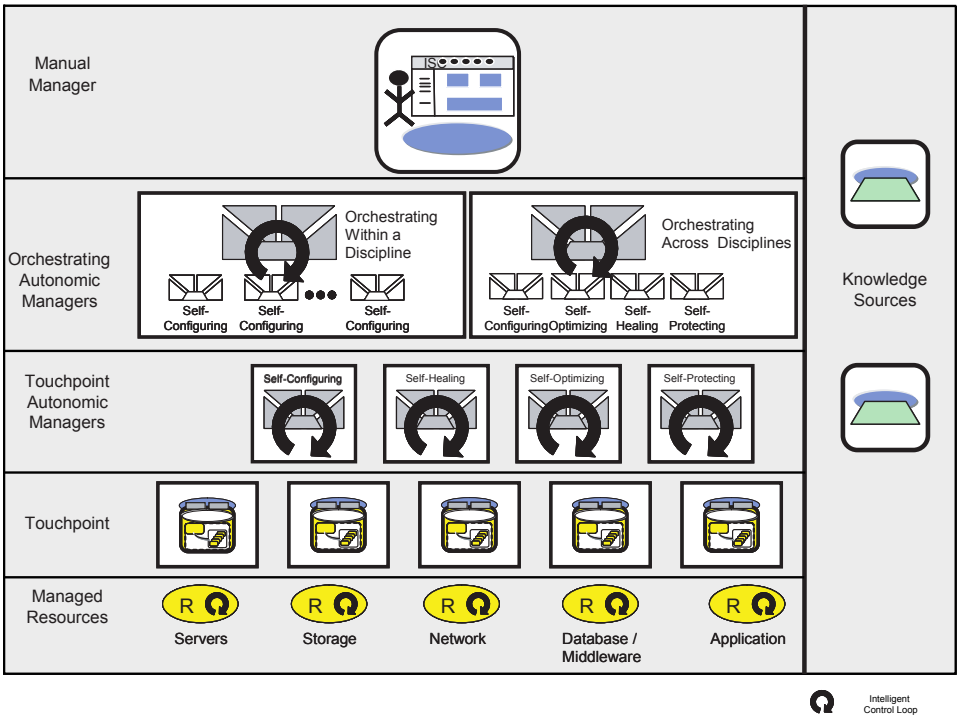


Figure 2.1: Autonomic Computing Reference Architecture based on [IBM06].

MAPE-loop for short — consisting of the following four steps.

- **Monitor module:** gathers information from the underlying system and possibly the system's environment in order to correlate and model complex situations, providing the subsequent computations of the control loop with the necessary data.
- **Analyze module:** examines the information previously gathered by the monitor module, and based on the identified symptoms draws plans on which further actions should be undertaken by the plan module.
- **Plan module:** constructs the sequence of actions that is needed to achieve goals and objectives
- **Execute module:** performs and controls the execution of such actions.

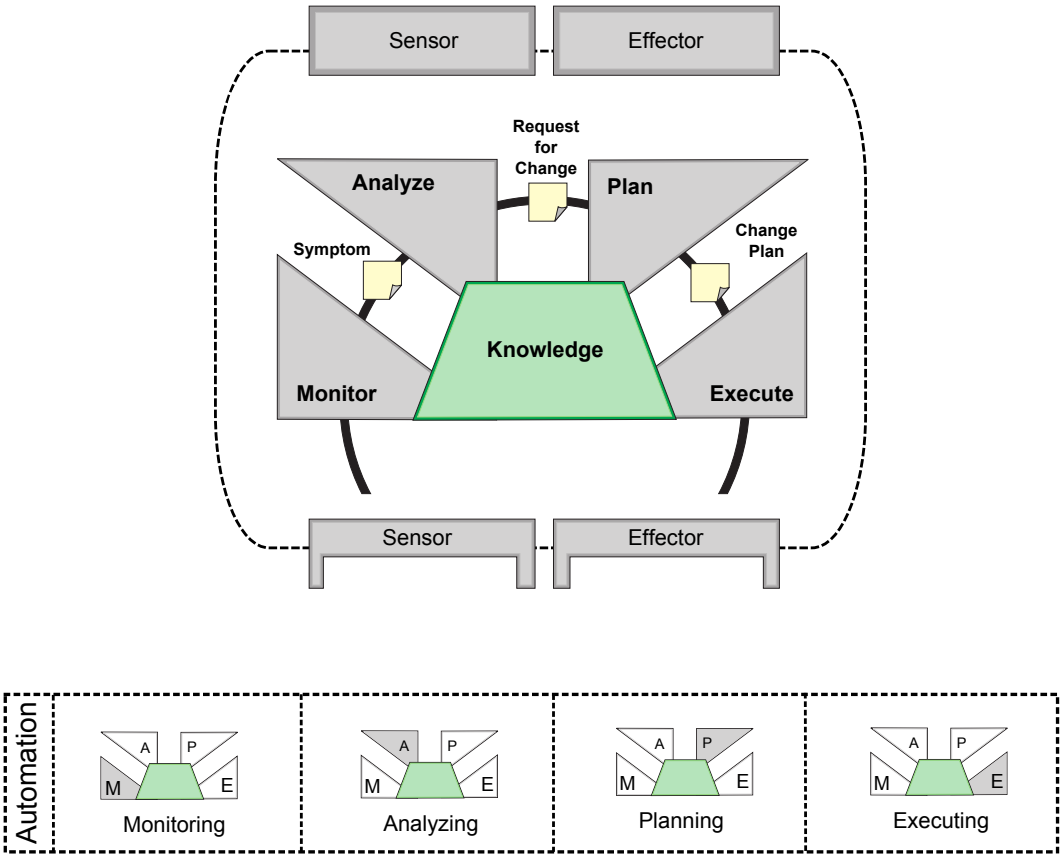


Figure 2.2: Functional representation of the MAPE-loop, based on [IBM06]. This is the most prominent way of forming feedback control loop in AC systems. It is composed of a monitoring module to collect data from underlying resources, an analysis module to detect and correlate complex situations, a reasoning module to plan the sequence of actions needed to achieve goals, and an execution module to control the execution of plans using a set of effectors. It is to note that knowledge stored of an autonomic manager can be also shared with other autonomic managers. This is important to support the collaborative decision making of managers in case to achieve a common goal.

2.2.2 Organic Computing

Closely related to the objectives of AC is the Organic Computing (OC) [ACE+03, Sch05] initiative — an academic-driven research funded by the DFG [2] — which mainly focuses on building interconnected distributed self-* system. In this sense an OC system is also characterized by self-* properties such as self-configuration, self-optimization, self-healing, self-protection, self-explaining, and context awareness. However, in contrast to the IBM'initiative which focuses basically on the self-management of data centers, an OC system has its focus on the development of generic and emergent properties for more smaller and more heterogeneous entities [JMC+06]. This means that an OC system is more confronted with unforeseeable situations in which the behavior of nodes can change quickly at runtime. To this end, a generic Observer/Controller architecture has been proposed for OC in [UMJ+06]. Its functioning is similar the MAPE-loop of AC with the sole exception of two major points: (1) It puts an additional flexibility with respect to where the control loop is placed in the system, either centralized, distributed or in multi-level structures. (2) It encapsulates many of the modules of MAPE but does not prescribe the stages of the control loop. Figure 2.3 gives an overview of the most important components of this architecture. The observer module mon-

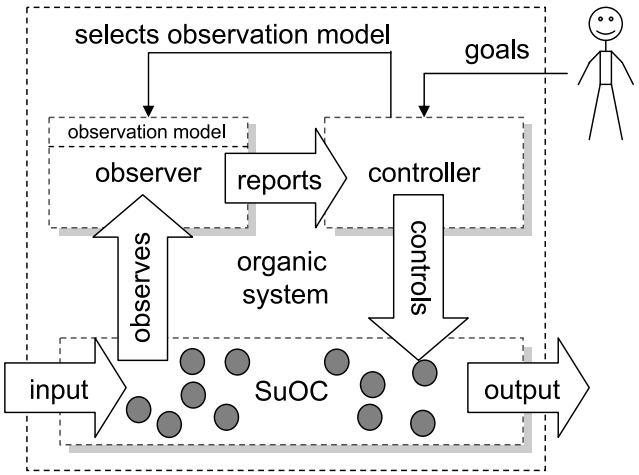
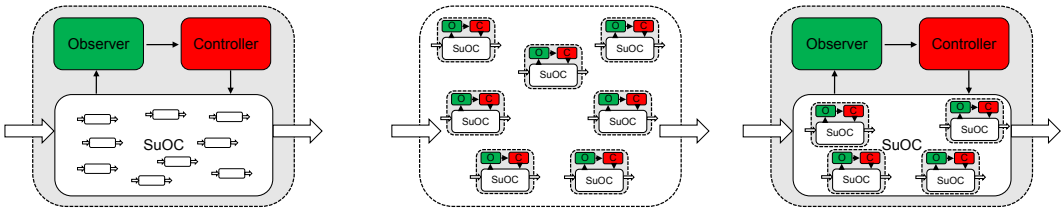


Figure 2.3: The Observer/Controller pattern in its basic form based on [UMJ+06].

itors the system and thus has the ability to detect unwanted behaviors. In such cases, the controller is used to specify corrective actions. The productive system that is managed by such an Observer/Controller pattern is called the system un-

[2] DFG stands for „die *Deutsche Forschungsgemeinschaft*“, which is a significant German research funding organization and the largest such organization in Germany.

der observation and control, or SuOC for short. There are different variants of Observer/Controller (O/C) pattern that can be customized depending on the application scenario and the complexity of the system. These variants are visualized in Figure 2.4 and realized on the following ways:



(a) Centralized O/C pattern (b) Decentralized O/C pattern (c) Multi-level O/C pattern

Figure 2.4: Variants of the generic O/C design pattern based on [CHMS08].

- **Centralized O/C pattern:** The centralized O/C pattern has only one control loop for the entire SuOC. The SuOC is regarded as a kind of black box where there is no need to distinguish between good and bad behaviors of the single system elements. In such cases, the observer monitors the data of the system as a whole and the controller takes corrective actions with respect to the entire SuOC. The type of this pattern is most often applied within weak self-organizing systems [SMSÇ+11], i.e., where the self-organization process is managed by a centralized point of planning and control.

- **Decentralized O/C pattern:** The distributed O/C pattern has an individual control loop for each system element. These elements are observed and controlled separately to enable the system to autonomously adapt their behavior to a given individual specification. Such a pattern would be appropriate for strong self-organizing systems, where there is no global or central knowledge for controlling the system [DMSGK05].

- **Multi-level O/C pattern:** In contrast to the decentralized approach, the multi-level O/C pattern proposes one control loop for each individual system element as well as one loop for the entire system. In this way holarchies [3] and system of systems can be built to provide system elements that are wholes and parts at the same time. Examples of systems using this pattern can be found in [FDMD15, MMV03].

[3] A holarchy is a system connection between holons, where a holon is both a part and a whole. For more details on this topic, please refer to [GB04]

2.2.3 Main Characteristics of Self-* Systems

Although the initiatives of AC and OC constitute different research areas, they converge in some similar characteristics as well. In the following, we summarize and clarify the main characteristics of distributed self-* systems:

1. **Homogeneity and heterogeneity:** System elements may have different behaviors with respect to their reliability, availability, functionality, and resource capability.
2. **Openness and scalability:** The number of system elements is not limited in the network and every one can join and leave the network at any time. Currently, there is an assumption of the benevolent behavior of system elements.
3. **Decentralization:** Self-* systems have often a decentralized structure in both senses of locality and control. The system elements are distributed among the network and are able to communicate with each other by using message passing.
4. **Adaptivity:** Self-* systems are inherently adaptive and capable of doing the adaptation necessary without manual guidance.
5. **Self-management**: The system itself decides how to manage objectives using a multitude of distributed self-* properties such as:
   - Self-configuration: Automatic configuration of system elements
   - Self-optimization: Automatic monitoring and control of resources in order to provide an optimal functioning regarding the defined requirements.
   - Self-healing: Automatic recognition and correction of failures
   - Self-protection: Automatic protection from arbitrary threats.
6. **Heterogeneous goals:** Every system element can have its own goal, which is part of the overall system goal.
7. **Flexibility:** The goals and objectives are not hard-coded and thus have the ability to be changed at runtime.
8. **Robustness:** Self-* system are designed to provide robustness against unexpected disturbances, especially those that have not been recognized at runtime due to emergent phenomena.

9. **Fault-Tolerance:** Self-* system should continue performing services after the occurrence of failures. The types of failures considered in this work are crash failure, execution failure and reachability failure.

2.3 Related Systems

The self-* system regarded in this work is an OC system having the same characteristics as those introduced in 2.2.3. Thus, our developed approaches can also be applied to any kind of AC systems, but for the purposes of this thesis it is worthwhile to mention other related systems from the literature.

2.3.1 Control System Theory

An interesting area that deals with rating and regulating of the output of dynamic systems is the Control System Theory (CST) [Oga10, Bur01], originally comes from the interdisciplinary field of mathematics and engineering. The focus of the latter is set on studying the behavior of dynamical systems with inputs, and how their behavior is modified by control loops. In this sense, a dynamic system DS produces some kind of output O from a given input I. This output O is continuously rated and regulated to a predefined reference value R. If there is no difference between the reference value and the resulted output, then no measured error is produced and thus the process can terminate. Otherwise, the control loop is used to bring the current output closer to the reference value in the next iteration. Figure 2.5 gives an overview of this main concept. In the literature, there exist several categories

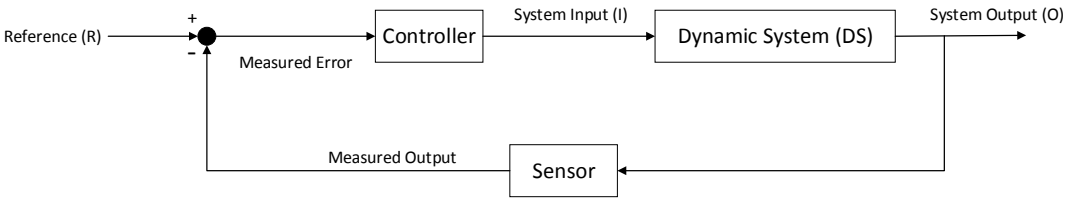


Figure 2.5: A simplified representation of a control loop illustrating that the measured error is received by a controller, which then either attenuates or amplifies the input signal to the dynamic system DS. Diagram is based on [ADH+08].

of control models for CST systems, such as *adaptive control*, *proportional-integral-derivative* and *model-predictive control* [JAR09], but as it goes beyond the scope of this work they are not further discussed here. Apart from the mentioned differences, CST systems also differ from the above introduced self-* systems in four

important points. First, the system behavior of CST systems is well-known, easy to understand and deterministic. This means the output can never be unforeseeable or unknown as soon as the input is known a priori. Second, the overall system configuration of CST systems is completely done at design time. Third, there is no network-communication between system elements in which every one can be responsible for its own single control loop. Finally, the reference value of CST systems has always to be known a priori, making the use of such systems unpractical for high complex environments.

2.3.2 Multi-Agent-Systems

The study of Multi-Agent Systems (MAS) [SLB09] focuses on technical systems in which several entities can interact with each other intelligently. These entities are called agents. They refer in most applications to autonomous constitutions such as humans, robots, embedded systems or even a mixture of different types of them. In general, these agents are used to model and solve problems in which the interaction between them can either be selfish or cooperative. In case of selfish behavior, the agents would pursue their own interests as in the free market economy example [KHS98]. In case of cooperative behavior, however, every agent would share a common goal — satisfying the overall system requirement — as in the example of ant colony [Par97]. The specific behavior of these agents depends on the system environment that is regarded. According to the roadmap given by [JSW98], a MAS system has the following characteristics:

- **Autonomy:** Each agent is autonomous or at least semi-autonomous that is able to perform asynchronous computations without the direct interventions of humans or others.
- **Local view:** Due to the system's complexity, none of agents have a global view of the entire system. They have to make decisions based on incomplete information of the environment where they are located.
- **Decentralized control:** None of agents have a global control over the entire system.
- **Decentralized data:** There is no central instance for storing data.

Due to these characteristics, MAS systems have attracted a profound admiration of researchers, which put forth novel techniques around the globe for investigating and implementing the deductive reasoning behavior of agents [BIP88, RK86].

Nonetheless, their popularity has quickly decreased with the years due to the controversial meaning of agents [FG97] as well as other multiple limitations and restrictions of the concept [Art94].

2.3.3 Cyber-Physical-Systems

There exists different definitions in the literature regarding Cyber-Physical-Systems (CPS) [SKB13, GB12, RLSS10]. In [GB12], we see a common definition that a CPS system consists of multiple entities that interact with each other to control different aspects of the physical system. In a CPS system, every entity has its own sensors, computation components, and actuators as well. Sensors are responsible to monitor the behavior of the external environment. They play a major role in providing input data for the computation components. The computation components perform then operations and generate output data, that is given to the actuators. The actuators make use of this data as basis to control some specific tasks in environment. The research field of CPS systems connect two different research areas, the research area of embedded real-time systems with the research area of digital networks [Lee08]. This is a nice property, that we can learn from, to make future self-* system real-time capable. However, one of the main disadvantages is its limitation to handle flexible goals as is known in self-* systems.

2.4 Role of Trust in this Thesis

Current self-* systems are based on the benevolence assumption that all entities in the system are trustworthy and interested to further the system goal. In open and heterogeneous systems, this benevolence assumption is unrealistic, since uncertainties about the entities' behavior have to be regarded. One solution to overcome this issue is *trust* [MD05, Mar94b]. Using appropriate trust techniques, entities in the system can have an information about which entities to cooperate with. This is fundamental to enhance the robustness of open distributed self-* systems which depend on a cooperation of autonomous entities [RAS+16]. A look at the literature reveals different ways to model trust in computer science, using either centralized or decentralized trust metrics. An example for a prominent centralized metric include the eBay Reputation Metric [RZSL06, GBS08], where all system entities store their experiences in a centralized repository and also request trust values from that repository. However, a decentralized trust approach stores and calculates the trust values strictly locally with no global control. The focus of attention in this thesis lies on decentralized approaches, since they are more suitable to be applied in distributed environments. Existing models based on such full decen-

tralized approaches can be found in [BFIK99, Jøs96, ARH97, Mar94a, MS12, TB06]. Even though there is some discrepancies between their construction, most of the approaches share one common model for building trust [MP09]. This is composed of five consecutive stage, as shown in Figure 2.6. In the first stage, the interactions

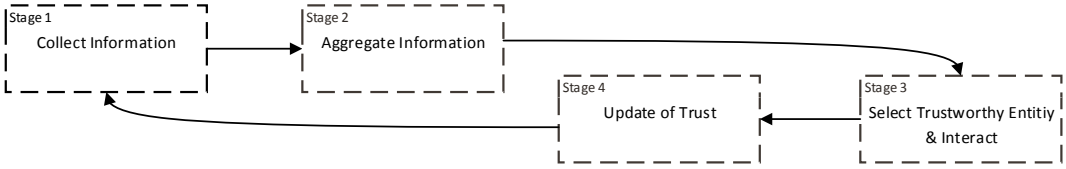


Figure 2.6: The generalized stages of building trust. Diagram is based on [MP09].

between system entities are observed, collected and stored in form of experiences. In the second stage, experiences are filtered and aggregated — depending on the specific context and the facet regarded in the system — to form a score that is given for every entity in the network. Then, the trustworthy entities are selected to get more responsibility for performing important tasks in the system than the untrustworthy ones. And finally according to the satisfaction obtained in the last step, a punishing or rewarding is carried out, adapting consequently the trust value deposited in the selected entity.

Research Position: This thesis is concerned with the use of trust to improve the robustness of distributed self-* systems. It is part of the research unit OC-Trust [4]. One of the key research questions that we have recognized within the scope of this project and of course within the community as well, is how to incorporate trust best possible in distributed self-* systems, both at design time and at runtime. This question was not easy to answer due to the fact that in such environments the structure of the system can highly be complex and volatile. Analyzing and studying different models of trust in [SKL+10] enable us to realize that there are some properties that must be taken into consideration when we speak about trust, because they correspond to the realities of many open distributed self-* systems. In the following we give an overview of these different properties:

- **Trust is a subjective property:** There is a consensus that trust is a highly subjective property. Every entity can have its own experiences with different entities in the system and make a personal trust value based on these unique

[4] OC-Trust: is an acronym of a research unit sponsored by the DFG that deals with the trustworthiness of Organic Computing systems.

experiences. Thus, the experience of two entities with a third same entity can vary enormously.

- **Trust is a multi-faceted property:** We see that there are several facets of trust that all contribute to the prediction of the entities'behavior. According to [SKL+10], these facets can be formulated as follows:
  - **Functional correctness:** The quality of a system to adhere to its functional specification under the condition that no unexpected disturbances occur in the system's environment.
  - **Safety:** The quality of a system to be free of the possibility to enter a state or to create an output that may impose harm to its users, the system itself or parts of it, or to its environment.
  - **Security:** The absence of possibilities to defect the system in ways that disclose private information, change or delete data without authorization, or to unlawfully assume the authority to act on behalf of others in the system.
  - **Reliability/Availability:** The quality of a system to remain available and reliable even under disturbances or partial failure for a specified period of time as measured quantitatively by means of guaranteed availability, mean-time between failures, or stochastically defined performance guarantees.
  - **Credibility:** The belief in the ability and willingness of a cooperation partner to participate in an interaction in a desirable manner. Also, the ability of a system to communicate with a user consistently and transparently.
  - **Usability:** The quality of a system to provide an interface to the user that can be used efficiently, effectively and satisfactorily that in particular incorporates consideration of user control, transparency and privacy.
- **Trust is a dynamic property:** We also see that trust can change over time especially within dynamic environments. Therefore, we distinguish between *initial trust* and *interaction-based trust*. The Initial trust is the trust value an entity gets for the first time when it interacts with another entity. This value can also be determined offline and will gradually be replaced by *interaction-based trust* if more interactions take place between the entities.
- **Trust is a context-sensitive property:** Entities have to adapt their trust in different contextual situations. For example, an entity e_1 may trust another

entity e_2 about the latter's behavior to solve a Problem p_1. However, this does not mean if e_1 wants to solve another Problem p_2 , it will trust e_2 to do it. This depends strongly on the context of the situation.

Due to the careful choice of the aforementioned properties and the system's complexity regarded in this thesis, we have chosen to model trust by means of different aspects, such as direct trust, reputation, confidence, and aggregation [KJMU13]. We refer the reader to [Kie14] for understanding the origin of these aspects, but technical details of them can also be found in Chapter 3. The direct trust serves as basis to assess the direct behavior of entities in order to provide consistent information when assigning or relocating services in changing environments. The reputation, confidence and aggregation aspects are designed to manage and query trust information between individual entities in the system.

2.5 Conclusions and Future Work

In this chapter, a background knowledge is provided that helps to better understand the main parts of this thesis. We introduced in Section 2.2 the current architectures of autonomous systems along with Organic and Autonomic Computing. Then, we pointed out their conceptual challenges, similarities, and differences more precisely. Based on that information, a set of characteristics is identified in Section 2.2.3 to generalize their most important properties. In Section 2.3, a general survey is given on other existing research areas related to self-* systems. The main focus thereby is set on Control System Theory, Multi-Agent-Systems, and Cyber-Physical-Systems, since they use very similar techniques to adapt to their environments as well. After analyzing them more in details, it becomes evident that none of the related approaches can substitute entirely the class of self-* systems. For instance, controllers in CST systems are not able to communicate in parallel way over MPI as it is possible in OC/AC and thus their application is limited only to some domains. Finally, we introduced in Section 2.4 our research position and explained how an ideal robust system could be achieved in open environments by combining both worlds of self-* system and trust.

3

System Architecture, From OCμ to TEM

An Overview of Trust Practices and Self-* Algorithms

Abstract. Open self-* systems of a very large scale – interconnecting several thousand of autonomous and heterogeneous entities – become increasingly complex in their organisational structures. This is due to the fact that such systems are typically restricted to a local view in the sense that they have no global instance, which can be responsible for controlling or managing the whole system. Therefore, new ways have to be found to develop and manage them. An essential aspect that has recently gained much attention in this kind of systems is the social concept of trust. Using appropriate trust mechanisms, entities in the system can have a clue about which entities to cooperate with. This is indispensable to improve the robustness of self-* systems, which depends on a cooperation of autonomous entities. The contributions of this chapter are trustworthy concepts and generic self-* algorithms with the ability to self-configure, self-optimize, and self-heal that work in a distributed manner and with no central control to ensure robustness. Some experimental results of our algorithms are reported to show the improvement that can be obtained compared with the baseline measurements.

3.1 Introduction

The proliferation of self-* systems capable of acting autonomously to achieve the overall system goal is already happening. Examples of such systems are Auto-

nomic and Organic Computing systems [Sch05, Hor01]. These are typically based on decentralized autonomous cooperation of system's entities and make use of a number of desirable self-* properties, such as the ability to self-configure, self-optimize, self-heal and self-protect in order to be manageable. The quality of their autonomy mostly depends on their ability to adapt their behavior in response to changes in their environment. At runtime, they should be able to trustworthy deal with situations not anticipated at design-time. One way to tackle trustworthiness issues is to enable humans to supervise the system and perform all trustworthy operations. However, this solution solves the problem only partially because the transfer of control to humans would also drastically decrease the system's autonomy, especially in the context of reactive environments. Therefore, new ways have to be found to ensure the trustworthiness of modern self-* systems by regarding different facets of trustworthiness. Such facets may concern, for example, reliability, credibility, availability, functional correctness and safety [SKL$^+$10]. In this chapter, we describe our efforts to develop a generic architecture that supports the trustworthy design of modern self-* systems. The baseline self-* system examined in this work is OCμ [RSK$^+$11], an Organic Computing middleware implemented in Java and based on a peer-to-peer network. All of its self-* properties were developed without trust involvement. We propose to incorporate a trustworthy self-* layer into the middleware to allow network entities to decide how far to cooperate with other entities. This information is used to maintain a trustworthy and robust configuration of the self-* properties in the face of untrustworthy entities. In general, this chapter offers as contribution the following aspects:

(i) some related works to relevant self-* middlewares originated from the field of Organic and Autonomic Computing systems (see Section 3.2),

(ii) a presentation of the OCμ middleware used in this thesis as baseline for our main results, as well as a description of the benevolence limitation that hinders the baseline system to perform well in hostile environments (see Section 3.3),

(iii) the novel architectural design of the OCμ middleware — transformed to TEM — to incorporate the trust-aware self-* properties (see Section3.4), and

(iv) a set of application case studies implemented based on TEM (see Section 3.5).

3.2 Related Work

This section presents relevant service-oriented self-* middlewares originated from the field of Organic and Autonomic Computing.

Lund et al. [LBB15] introduce an organic middleware called Artificial Hormone System (AHS) – providing self-* properties – to autonomously assign tasks to heterogeneous processing elements. The middleware makes use of different artificial hormones to find the best suitable processing element (PE) taking into account constraints such as the current PE workload and the task relationships. The following type of hormones exist: *eager value*, *suppressor* and *accelerators*. The eager hormone aims to determine the suitability of a PE to execute a specific task. The other hormones are responsible for reducing or increasing the eager value and thus by applying suppressor and accelerator respectively. Through the use of hormones, the AHS middleware implements the self-* properties: self-configuration in terms of finding an initial allocation of tasks by exchanging hormones, self-optimization by task migration when hormone levels change, and self-healing by autonomous task reassignment due to task or resource failure. Compared to our middleware, the AHS assumes the trustworthiness of PEs to perform well, i.e., all PEs are considered to be trustworthy to further the system goal.

CARISMA [NB09a] is a service-oriented middleware for hard real-time environments. It realizes self-configuration and self-optimization with the focus on real-time system capabilities. The services have real-time constraints, meaning that their correctness does not only depend on their computational results, but also on the time at which the results are delivered. For allocating services, CARISMA makes use of an auction mechanism based on the Contract Net Protocol [Smi80] to decide whether a service can be contracted and which quality can be achieved. Afterwards, the whole system is optimized by re-contracting the services to other nodes with better resource availability. In contrast to CARISMA, our middleware focuses on the general applicability of system's service that are not restricted by real-time constraints. Additionally, CARISMA does not provide a differentiation between the importance levels of services. This differentiation is necessary to allocate the most important services only on trustworthy nodes and thus to increase the robustness of the system.

FraSCAti [SMR+12] is a service-oriented middleware supporting the Autonomic Computing principles. It exhibits self-* properties to add new services at runtime or to remove existing ones. The self-* properties are obtained by applying the MAPE (monitoring, analysis, planning, and execution) control loop of Autonomic Computing. The monitoring phase is responsible for collecting, aggregating and filtering the information collected from the services and the middleware itself. The gathered information is examined in the analysis phase. If the examination reveals a need to adapt the placement of services, a new plan is created which is then realized in the execution phase. In contrast to our middleware which supports trust

in the development of the self-* properties, FraSCAti expects the benevolence of device nodes limiting its usefulness in open environments.

An organic middleware for building self-organizing smart camera systems is presented in [HWHMS08, JSS+13] and has been extended in [TJHH13] to provide support for cloud services. The middleware consists of a number of cameras which are able to collaborate together to detect intruders in non-public areas. At runtime, each camera can adapt its position view while keeping other cameras'positions. The self-healing property is used to maintain the whole tracking system stable even under failure of a single camera or a loss of connectivity. The overall result afterwards is achieved by merging the sub-results of each camera. The general assumption here is that cameras always voluntarily cooperate to realize the common goal. This assumption hinders the middleware to be used in hostile environments, in contrast to our work. Parts of the content of this chapter have been published by the author in the following book chapter and technical report:

- **[MU16a]:** Nizar Msadek and Theo Ungerer. *Trust as Important Factor for Building Robust Self-x Systems*. Book chapter in Trustworthy Open Self-Organising Systems, Autonomic Systems series, Vol. 7, 2016, Springer International Publishing, pp 153-183, `http://www.springer.com/de/book/9783319291994` ISBN: 978-3-319-29199-4

- **[ASM+13]:** Gerrit Anders, Florian Siefert, Nizar Msadek, Rolf Kiefhaber, Oliver Kosak, Wolfgang Reif, and Theo Ungerer. TEMAS — A Trust-Enabling Multi-Agent System for Open Environments enabling multi-agent system for open environments. Technical report, University Augsburg, 2013.

3.3 The Baseline OCμ Architecture

The baseline self-* system considered in this work is OCμ [RSK+11], a middleware for Organic Computing systems. The OCμ middleware was developed in the German Research Foundation (DFG) priority program "Organic Computing" at the University of Augsburg and is comparable to other state of the art distributed, service-oriented middleware architectures. But it has the advantage that it implements several self-* properties and thus has the ability to be slightly extended to provide the desired trustworthy self-* layer. The OCμ middleware consists of a collection of heterogeneous devices – called nodes for short – with diverse capabilities of computing power, memory space and energy supply. These devices interact with each other using message passing. An overview of a single OCμ node is illustrated in Figure 3.1. It is composed of five main parts: the transport connector, the

message dispatcher, the service proxy, the service interface, and the services which are explained in the following.

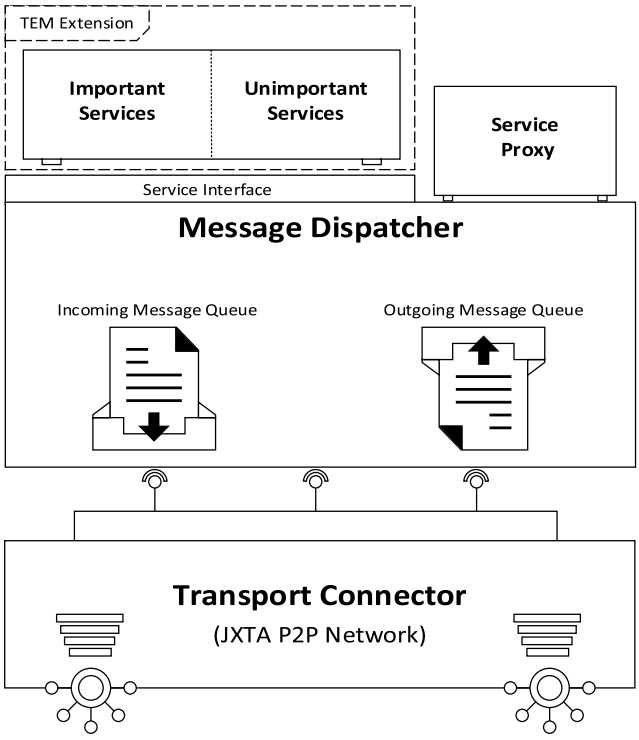


Figure 3.1: Structure of the baseline OCμ node depicting its most relevant parts: the transport connector, the message dispatcher, the service proxy, the service interface and the services.

- **Transport Connector:** The communication system used in the middleware is modeled similar to [DL88]. Each OCμ node p has a buffered transport connector enabling it to connect fast and reliably with other OCμ nodes. In the baseline implementation, the two following communication primitives are used:
  - **Receive**: removes existing messages from p's buffer and delivers the messages to the message dispatcher of p.
  - **Send**(m, q)**:** Sends a message m over p's transport connector and places it in the buffer of q.

  The protocol used for the current implementation of the transport connector is JXTA[1]. This can be replaced or extended with any other communication

[1]JXTA: Open source peer-to-peer protocol specification begun by Sun Microsystems in 2001 – [Ac-

protocol, since it is transparent to the rest of the middleware.

- **Message Dispatcher:** The message dispatcher handles the message delivery between the services in the middleware. It offers the services the functionality to send messages and register themselves as listener for specified types of messages. With this functionality it is also possible for a service to register itself for different types of messages. In such a case, the service will be informed whenever a message with one of the registered types is received.

- **Service Proxy:** A service proxy is used to forward messages for a service that was recently moved to another node. During the service transfer, the service proxy stores the incoming messages, it then forwards them as soon as the service becomes available at the new position node. The life span of a service proxy is predefined at runtime by its corresponding service, such that it dies after that time.

- **Service Interface:** The service interface is the connector between the middleware and a service. Each service has to implement this logical interface to bind itself to the middleware. The interface provides all required methods to send and receive messages via the message dispatcher to the services.

- **Services:** The considered middleware is based on the assumption that applications are composed of services, which are distributed to the nodes of the network. These services are implemented without priority consideration in OCμ. Later in TEM, an extension has been provided for categorizing services. We distinguish between two kinds of services, namely important services and unimportant services. Important services are those which are necessary for the functionality of the entire system. However, unimportant services are those which only have a low negative effect on the entire system if they fail.

A crucial part of the OCμ middleware is to investigate decentralized solutions to self-configuration, self-optimization and self-healing [SRK+11]. These self-* properties were developed independently without trust guidance. In the following, a general description of their functional parts is given:

- **Self-configuration [TKU06]:** The regarded applications produce a set of services that are independent of each other. These services are initially distributed to available nodes in the network through the self-configuration process. If, for example, a new node joins the system, it will configure itself au-

cessed: December 16, 2015] – `http://jxta.kenai.com`.

tonomously in the middleware such that the overall resource utilization is as good as possible.

- **Self-optimization [TPSU07]:** The self-optimization process enables the system to autonomously reach an optimized state. This optimization might concern the runtime allocation of services on nodes to ensure a uniform distribution of load.
- **Self-healing [Sat08]:** The self-healing aims to ensure a valid state of the system even in the presence of failure. If, for instance, a node fails, the system must be able to detect its failure using the runtime monitoring approach presented in [SPTU08] and then to restart all of its services on other available nodes.

The baseline middleware introduced so far does not consider the trust behavior of the system during runtime. It is based on the benevolence assumption that all participating nodes want to help each other whenever possible to further the system goal. Because of this assumption, nodes were considered to be always trustworthy and the self-* properties were developed without regarding the node's trustworthiness. However, in open and heterogeneous systems where nodes can enter and leave the system at any point in time, this benevolence assumption has to be dropped, since nodes might behave untrustworthy and try to exploit the system. This introduces a level of uncertainty in the middleware that has been largely neglected so far.

3.4 The Trust-Enabling Middleware TEM

The integration of trust to deal with uncertainty gives OCμ the ability to better adapt to changes in the environment. The approaches and techniques proposed in this section are technological and based on the notion of trust to enable the creation of more robust self-* properties.

3.4.1 General Overview

The extension of the OCμ proposed in this section aims to incorporate trust into the basic self-* properties. For this, the distributed Observer/Controller architecture suggested by Richter et al. in [RMB+06] is used and refined by providing trust guidance. Figure 3.2 gives an overview of the proposed TEM concept. The baseline OCμ node is enhanced by a trustworthy self-* layer providing a feedback control loop to observe and control the behavior of OCμ. It includes an observer component, the functional element responsible for monitoring the trust behavior of the

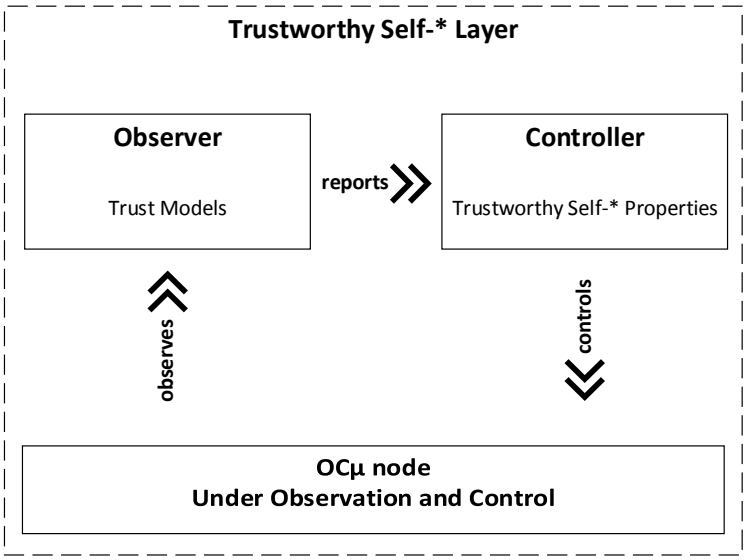


Figure 3.2: The generic Observer/Controller architecture used for the establishment of the trustworthy self-* layer. The observer incorporates trust models and the controller is composed of trust-aware self-* properties. Communication between observer and controller is based on the feedback control loop that OCμ nodes provide.

baseline system. This is performed by using trust metrics enabling the generation of data about the trustworthiness of the system's entities. The collected data is used by the controller as indicator to make decisions that will influence the future course of the self-* properties. The following section describes the functional parts of the observer and controller components.

3.4.2 The Trust Observer

The main objective of the observer is to monitor the current behavior of nodes in the system and to calculate trust data from this information. This trust data is used by the controller to guide the overall system goal in a trustworthy way by applying the self-* properties. The observation process mainly consists of the following three steps: monitoring, transformation, and trust interpretation, as shown in Figure 3.3. In the monitoring step, a distributed strategy is needed to allow nodes to autonomously determine who is monitoring whom. In order to do this and to be at least scalable, we make use in this part of work of our former developed self-monitoring approach [SU08] as basis for enhancing nodes with the ability to collect raw data in the system. These raw data represent experiences that nodes have made with their interaction partners in a specific context and per trust facet. They

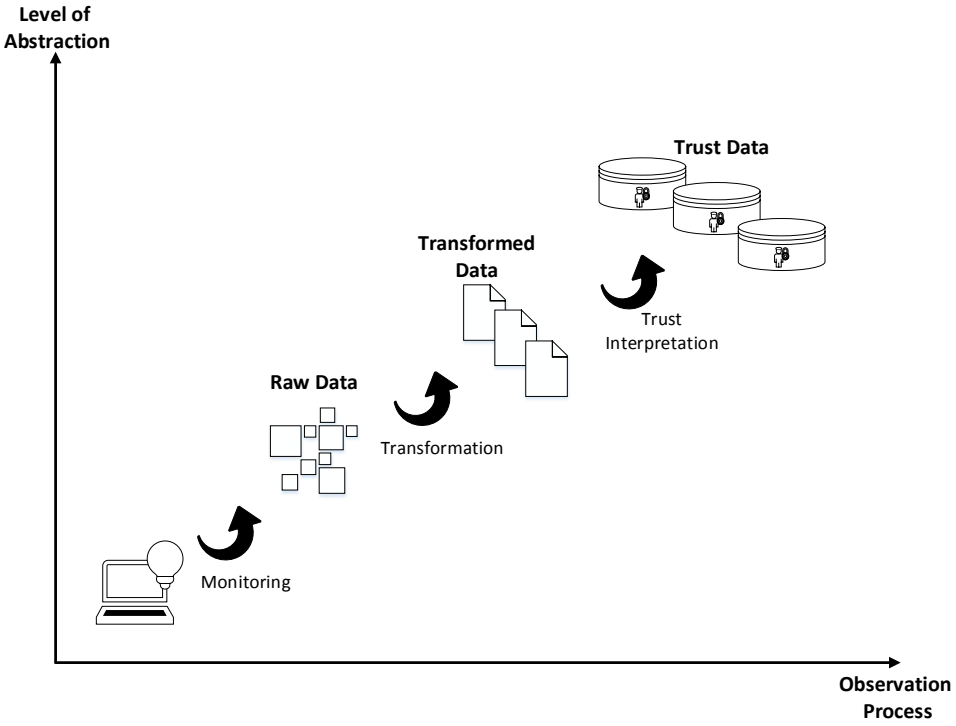


Figure 3.3: Illustration showing the different levels of abstraction of the observation process starting from monitoring, to transformation, to the final interpretation of the trust data. Please note that only the most interesting parts of the process are presented, due to space limitations.

contain all relevant information about the interaction partner such as message delays and loss as well as its ability to perform services at the right time. The raw data are stored in a distributed log file for every loop of observation. Then, they are transformed into a valid format that makes them easily accessible to the interpreter. Finally, the interpreter uses these transformed data to estimate the trustworthiness of a node. The metrics used by the interpreter to calculate trust data are named *trust metrics* [Kie14]. These metrics have been developed during the two first phases of the OC-Trust project and integrated in TEM. The metrics consist of the following presented aspects.

Direct Trust

The algorithm used by the observer to calculate the direct trust of a node is called the *Delayed-Ack* [KSS+10]. The Delayed-Ack algorithm covers the reliability aspect of trust as a facet and measures trust by observing the message flow between nodes. More precisely, it requires that each sent message is being acknowledged. Thus the lost of each message is determined, resulting in a negative experience (represented by 0) for lost messages and a positive experience (represented by 1) for acknowledged messages. All these experiences are stored for each participating node. The output is a direct trust value $t_{dt}(n_i, n_j)$ within $[0, 1]$ calculated by taking the mean or weighted mean of past experiences. $t_{dt}(n_i, n_j) = 0$ means n_i does not directly trust n_j at all while a value of 1 stands for a whole trust.

Confidence

In addition, a metric is used to evaluate the confidence of the own direct trust value of a node. This is called the *confidence metric* [KAS+12] and aims at describing how reliable the direct trust value is. The higher the confidence is, the more certain one can be that the trust value matches the actual behavior of an interaction partner. The confidence value is split in three parts:

- **Number Confidence:** The more experiences exist, the higher is the confidence, up to a threshold τ_n. Figure 3.4a shows details of this function. If the number of experiences $|X|$ is greater or equal to τ_n then the *number confidence* $c_n(|X|)$ is 1.
- **Age Confidence:** Every experience x is rated regarding its actuality a_x. The resulting rating $r(a_x)$ describes how recent or outdated the experience is (see Figure3.4b). The *age confidence* is higher if the experiences were made more recently. Two thresholds, τ_o and τ_r, are defined for this rating function: An experience older than τ_o counts as outdated and its age rating is set to 0. If an

experience is newer than the threshold τ_r, then it counts as a recent experience and its age rating is therefore set to 1. From τ_r to τ_o, the age rating is gradually decreasing. The total age confidence $c_a(X)$ is the mean of all ratings (see Equation 3.1).

$$c_a(X) = \frac{\sum_{x \in X} r(a_x)}{|X|} \tag{3.1}$$

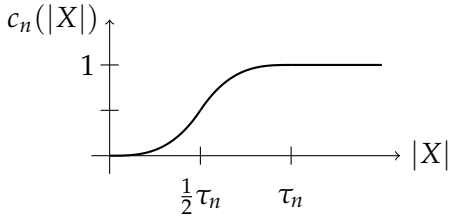
- **Variance Confidence:** It evaluates the variance of the experience values v_X. The more the values fluctuate, the lower is the *variance confidence* $c_v(v_X)$. If the experiences have 0 variance, i.e., the experiences are rated exactly the same, the variance confidence is 1. It decreases with increasing variance (see Figure 3.4c).

Confidence Value: The total confidence $c(X)$ is then calculated by a weighted mean of the three parts, as seen in Equation 3.2. w_n denotes the weight for the number confidence, w_a the weight for the age confidence and w_v the weight for the variance confidence. We assume $w_n, w_a, w_v \geq 0$ and $w_n + w_a + w_v > 0$.

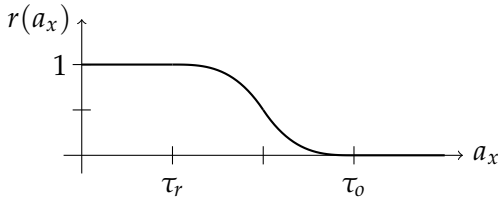
$$c(X) = \frac{w_n \cdot c_n(|X|) + w_a \cdot c_a(X) + w_v \cdot c_v(v_X)}{w_n + w_a + w_v} \tag{3.2}$$

Reputation

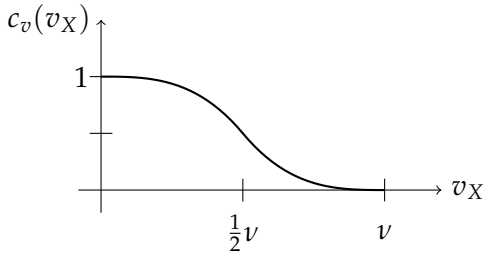
As reputation metric, the *neighbor-trust metric* introduced in [KHS+11] is used. The metric is based upon a weighted mean value of the direct trust values of all other nodes that had direct interactions with the node, the so-called *neighbor*s. The weights represent the truthfulness of neighbors regarding their reputation data. A high weight indicates a neighbor whose reputation data correlates with direct experiences of oneself, whereas a low weight stands for a neighbor whose reputation data differs a lot from the own experiences. To achieve this, two thresholds are defined for the reputation metric: τ defines the positive area, where reputation and direct trust are similar enough to increase the weight, the larger $\tau^* (\tau^* \geq \tau)$ denotes the negative area, where reputation and direct trust are too far apart which will reduce the weight. If the difference between reputation and direct weight is greater than τ^*, then the weight is decreased by a maximum of θ. Similarly, the weight is increased by a maximum of θ. Therefore, upcoming reputation information from a neighbor will be rated up or down depending on the information the neighbor gave so far. A node will then only listen to other neighbors whose experiences are similar to its own. Since the weight gets adjusted per interaction, the reputation has to start with an initial value, which is defined as r_s. Figure 3.5 depicts the func-



(a) Illustration of the number confidence function $c_n(X)$ assuming that a certain number of experiences τ_n is sufficient to derive an accurate trust value. The more experiences with an interaction partner were made, the more confidence in the trust value.



(b) The age confidence function $r(a_x)$ aims at calculating a low rating value for a quite outdated experience and high rating value for a quite recent experience.]



(c) The variance function $c_v(v_X)$ is used to indicate the behavioral changes of a node. A confidence of 1 equals 0 variance. Vice versa, a confidence of 0 equals 1 variance.

Figure 3.4: The three parts of the confidence value.

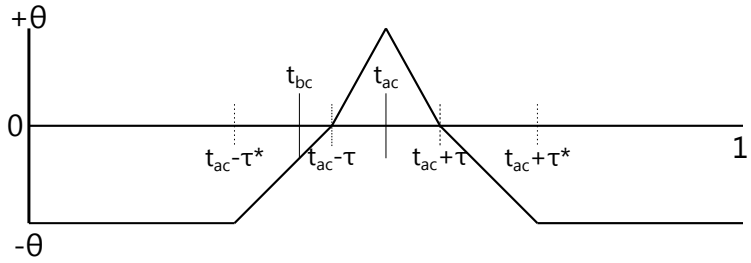


Figure 3.5: A graphic representation of the neighbor-trust metric. t_{ac} denotes the direct trust a node a has about another node c and t_{bc} the reputation information of node b about node c. In this example, the weight a has about b would be reduced, because both values differ by more than τ.

tion to calculate the weight adjustment after each interaction. t_{ac} denotes the direct trust a node a has about another node c and t_{bc} the reputation information of node b about node c. In the figure the weight a has about b would be reduced, because both values differ by more than τ.

Aggregation

When all the aforementioned values are obtained, a total trust value t_{total} based on the direct trust t_{dt} and reputation t_r values can be calculated using confidence $w_c(c(X))$ to weigh the both parts against each other. This t_{total} value is calculated with Equation 3.3.

$$t_{total} = w_c(c(X)) \cdot t_{dt} + (1 - w_c(c(X))) \cdot t_r \tag{3.3}$$

The higher the confidence, the higher is $w_c(c(X))$ and therefore the weight of the direct trust data in total. The formula to calculate $w_c(c(X))$ is based on the function depicted in Figure 3.6. This function is enclosed in two thresholds τ_{cl} and τ_{ch}. Outside these thresholds, the function is constant with extreme values; in between them monotonically increasing with near linear slope in the middle at $\frac{\tau_{ch}-\tau_{cl}}{2}$. Near the thresholds the slope is low. This function is based on the consideration that a small step over a threshold should only result in a small change in value. The domain is restricted to $[0, 1]$, because valid confidence values must be in this interval. The result of the function, i.e., the co-domain, is also $[0, 1]$ representing the weight $w_c(c(X))$ for the aggregation function by means of Equation3.4.

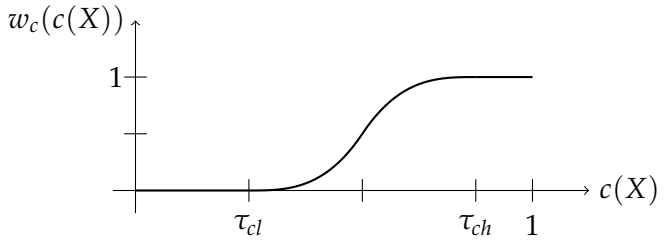


Figure 3.6: A graphic representation of the function $w_c(c(X))$. The higher the confidence, the higher $w_c(c(X))$ and therefore the higher the influence direct trust has over reputation.

$$w_c(c(X)) = \begin{cases} 0 & \text{if } c(X) < \tau_{cl} \\ 4\left(\frac{c(X)-\tau_{cl}}{\tau_{ch}-\tau_{cl}}\right)^3 & \text{if } \tau_{cl} \leq c(X) \leq \tau_{cl} + \frac{1}{2}(\tau_{ch} - \tau_{cl}) \\ 4\left(\frac{c(X)-\tau_{ch}}{\tau_{ch}-\tau_{cl}}\right)^3 + 1 & \text{if } \tau_{cl} + \frac{1}{2}(\tau_{ch} - \tau_{cl}) < c(X) \leq \tau_{ch} \\ 1 & \text{if } \tau_{ch} < c(X) \end{cases} \tag{3.4}$$

Figure 3.7 summarizes the main hypothesized trust metrics used for the calculation of total trust. Experiments in [KJMU13] attest that integrating such metrics into our middleware results in a better estimation of the real hidden trust value of an interaction node with increasing number of its interactions. For more information about the implementation details, please refer to [ASM+13].

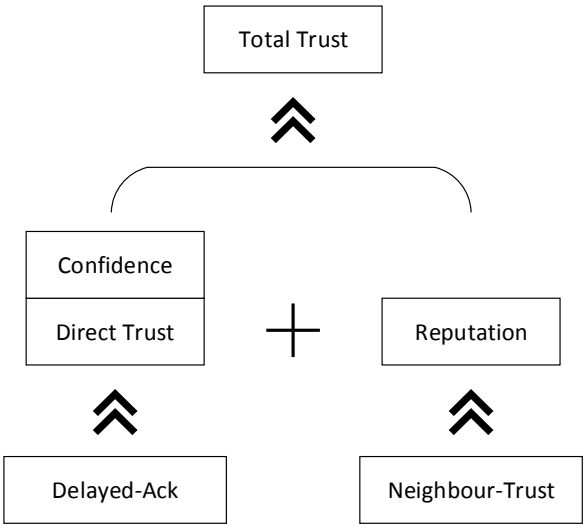


Figure 3.7: Illustration showing how direct trust, reputation and confidence are aggregated to form the total trust.

3.4.3 The Trust-Enhanced Self-* Controller

The aim of the controller is to guide and control the self-organization process between nodes. To make trustworthy control decisions, it uses the trust data received from the observer and affects the global system by influencing the execution rules of self-* properties. The current implementation suffers from the drawback that the self-* properties are not designed to incorporate trust decisions in their actual executions. They assume the benevolence assumption of nodes at all time and thus cannot be applied for open systems. Details on this assumption were given

in Section3.3. In this work, we abandon this benevolence assumption and instead provide a new trustworthy design of the baseline self-* properties allowing them to operate robustly even in open and hostile environments.

The Trust-Aware Self-Configuration

The capability of self-configuration in open distributed environments comprises the abilities (1) to perform an initial and trust-based distribution of services on nodes, (2) to cope with the problem of scalability, and (3) to allow a reconfiguration of the system during runtime due to self-optimization or self-healing demands.

- **Regarding (1):** There are many sophisticated approaches to deal with the initial distribution of services on nodes, either to achieve good load balancing or to minimize energy consumption. An approach that has become a standard by FIPA[2] is the Contract Net Protocol [Smi80]. It consists of finding an agent that is the most suitable to provide a service. This approach is often adapted and applied in many application domains, for example, manufacturing systems [HC09], resource allocation in grids and sensor web environments [KB09, GG08], as well as in hospitals [DGB03], electronic marketplaces [DKRA00] or power distribution network restoration [KHS+09]. It is a generic protocol [Boz08] and thus provides an excellent basis for developing self-configuring systems. However, it is limited in some issues and has some shortcomings if the setting for service assignment is more complicated. For example trust limitation in the service assignment – some of nodes are more trustworthy to do important services while others are less trustworthy and should focus only on the processing of unimportant services. Helping to develop these trust enhancements was the aim of our self-configuration research. The outcome of this investigation is an approach presented in Chapter 4.3 which is based on the Contract Net Protocol. Our approach aims on the one hand to equally distribute the load of services on nodes as in a typical load balancing scenario and on the other hand to assign services with different importance levels to nodes so that the more important services are assigned to more trustworthy nodes. Similar to [Smi80], nodes in our system can act as a manager or contractor. A *manager* is responsible for assigning services. A *contractor* is responsible for the actual execution of the service. Figure 3.8 depicts how managers and contractors can participate in the distribution phase of the self-configuration approach. When the assignment

[2]FIPA: Foundation for Intelligent Physical Agents – [Accessed: October 29, 2015] – `http://www.fipa.org/specs/fipa00029/`

process starts, managers announce the list of services to the contractors. Contractors evaluate these services and submit bids on those for which they are suited. Then, the managers evaluate the bids. In the basic Contact Net Protocol, the parameter characterizing this evaluation is the workload. Generally, the lower the workload of a node is, the more it is considered to be appropriate to receive the service. Our enhancement improves the awarding part by including trust, to enable that more trustworthy contractors always have a higher chance to receive services than less trustworthy contractors. Finally, the result of the service assignment is communicated to the contractors that submitted a bid. Evaluation results within our middleware (see Chapter 4.3)

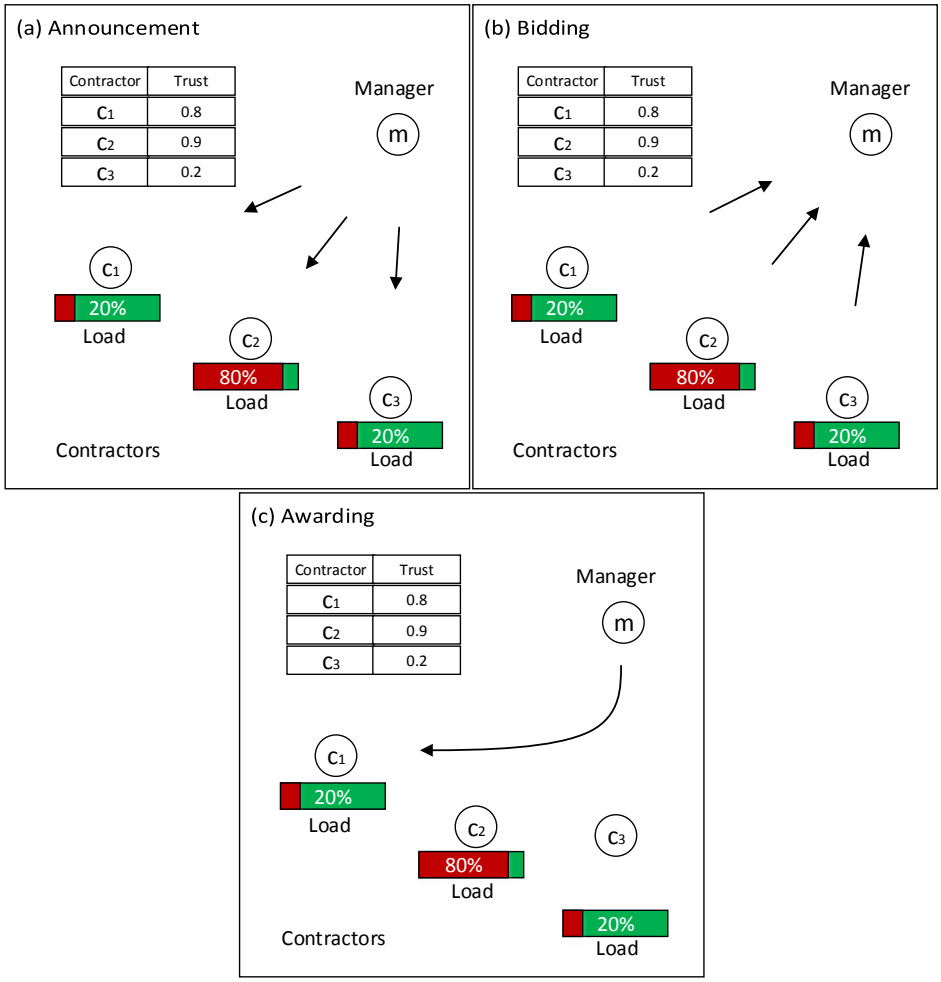


Figure 3.8: An overview of the self-configuration process showing the interactions between the manager and its contractors. Please note that each contractor in the network can be, at the same time and for different services, a manager of other contractors.

show that the proposed self-configuration algorithm indeed provides better performance than the baseline Contract Net Protocol. The trust variation that is used to improve the availability of important services performs much better in all cases than the baseline algorithm, which underlines the effectiveness of our approach.

- **Regarding (2):** In open environments, the issue of scalability is of particular importance for any self-configuring system. The self-configuration presented until now does not cover this issue. It was designed only for the sequential assignment of services on nodes and thus provides solutions that are not realistic to be applied in environments with many managers. To get a better comprehension of that problem, please consider the following example: Assuming two managers m_1, m_2 and one contractor c_1. m_1 is responsible for assigning service s_1 and m_2 is responsible for assigning service s_2. Then, consider the following sequence of operations that are listed in Table 3.1.

Manager m_1	**Contractor** c	**Manager** m_2
Sends an announcement to c for a service s_1		Sends an announcement to c for a service s_2
	evaluates the given services with respect to its workload and sends a bid to m_1 and m_2	
Sends an award message to c informing it to be the most appropriate		
		Chooses c to award him the contract for s_2, **while the latter submitted bid has recently become obsolete!**

Table 3.1: Simplified example run of the self-configuration process that can exhibit a race condition between managers m_1 and m_2.

Contractor c receives both services s_1 and s_2, as expected. However, if both managers m_1 and m_2 perform their negotiations in parallel and without coordination, the outcome of the assignment could be wrong. Despite not having enough resources, contractor c uses the same bid value submitted as before to receive the service s_2. Because of this race condition, we need to incor-

porate coordination strategies into our self-configuration algorithm in order to further improve its scalability performance. Investigating and developing such strategies was the aim of our work in Chapter 4.5 and the outcome of this research is a simultaneous self-configuration algorithm which gives managers in our system the possibility to perform several distribution phases at the same time. To quantify our approach, evaluations have been conducted. The evaluation results presented in Chapter4.5 show that the simultaneous self-configuration attests an excellent time performance to assign services on nodes than the sequential approach. At least 50% of the self-configuration time improvement was achieved and thus only for the context of two managers. However, the drawback of our approach is that it produces message overhead to coordinate the managers. But, this overhead is not excessive, in most cases lower than 1% compared to the sequential approach, and is thus considered to be acceptable in use by our middleware.

- **Regarding (3):** Reconfiguration is a main characteristic of modern distributed self-configuring systems. Managers have to assess at runtime whether contractors are operating correctly or not. Fault-tolerant techniques applied to our self-configuration algorithm are given in Chapter 4.4. If one of the contractors failed, managers detect the failed node and trigger a reconfiguration in the system to re-establish the balance between nodes again. The reconfiguration is applied even well in situations in which nodes join the system. This can occur at any time during the self-optimization process. Managers identify the entering of nodes and reconfigure themselves to regain an acceptable assignment state in the system.

The Trust-Aware Self-Optimization

Self-* systems should be able to dynamically adapt their behavior in response to changes in their environment. At runtime, they should have the ability to deal with situations not anticipated at design time, since not every situation can be considered when designing the system. After the initial service distribution that is given using the self-configuration process, nodes must be able to constantly observe their current resource consumptions as well as the trustworthiness of nodes they are cooperating with, identify unacceptable situations and reconfigure themselves to regain an acceptable state. Therefore, in Chapter 5 a self-optimization algorithm is presented to optimize the allocation of services on nodes during runtime. The algorithm does not only consider pure load-balancing but also takes trust into account to improve the assignment of important services to trustworthy nodes. More precisely, it uses different optimization strategies to determine whether a service

should be transferred to another node or not. Figure 3.9 illustrates how the self-

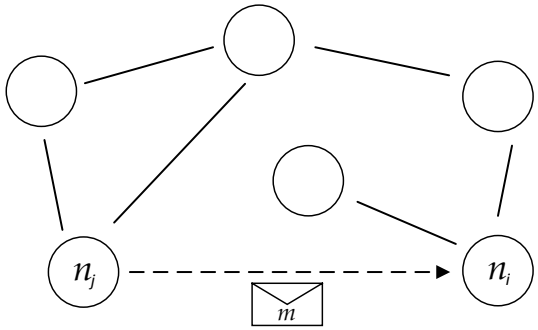


Figure 3.9: Cooperative self-optimization in the TEM middleware. The optimization process is initiated by an application message going from node n_j to another node n_i. This message contains – as piggy-back – all relevant information allowing both nodes n_j and n_i to optimize their current states in the system at runtime.

optimization property works in our system, using a simple example of just two nodes. Suppose that node n_j sends an application message to another node n_i at a certain point during runtime. It appends onto the outgoing message (a) its recently observed trust behavior of node n_i, (b) its current workload and (c) some additional information (i.e., importance level and consumption) about services which are running on it. Based on this information, node n_i decides which of the following optimization strategies should be performed (given in Table 3.2):

Workload Similar	**Trust Similar**	**Optimization strategy**
True	False	Trust Optimization
False	True	Load Optimization
False	False	Trust and Load Optimization
True	True	No Optimization

Table 3.2: Type of strategies the nodes can use to optimize their current states in the system.

- **Trust Optimization:** The trust optimization strategy is used in situations in which the workload of both nodes is similar but their trust values differ, as illustrated in Figure 3.10. Important services are relocated to the more trustworthy node and unimportant services to the less trustworthy node. The

workload balance, however, should still be maintained.

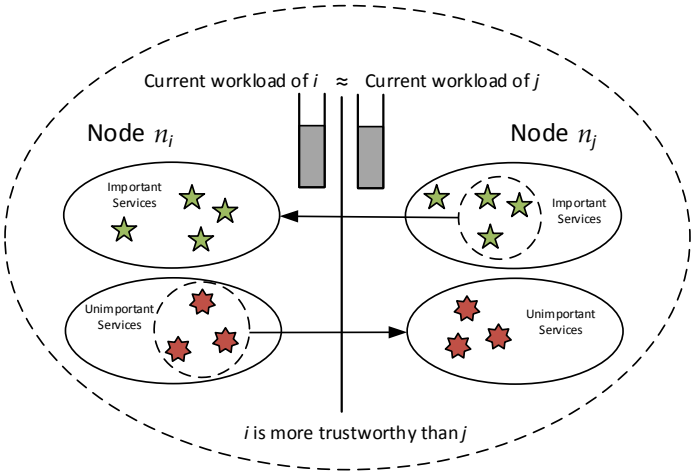


Figure 3.10: The trust strategy is depicted in a simplified form to optimize the state of nodes by relocating the assignment of their services at runtime. Please note that important services are represented by the green stars, whereas the unimportant services are depicted with red starlets.

- **Load Optimization:** The second strategy is the load optimization strategy, presented in Figure 3.11. This strategy aims at finding a pure load balancing between nodes. Since n_i and n_j are equally trustworthy with respect to a certain threshold, there is no need to consider trust by the relocation of services.

- **Trust and Load Optimization:** The trust and load optimization strategy allows for providing workload balancing with additional consideration of the services' priority to avoid hosting important services on untrustworthy nodes (see Figure 3.12).

- **No Optimization:** Finally, the *No Optimization* strategy is used to ensure termination when no further optimization can take place, e.g., if both nodes n_i and n_j are well optimized in terms of trust and workload, as shown in Figure 3.13. However, this termination is determined only locally. A global termination is reached if the system as a whole becomes optimized.

Experiments have been conducted based on simulations in Chapter 5.7 to evaluate the effectiveness of the introduced trust-aware self-optimization algorithm. The

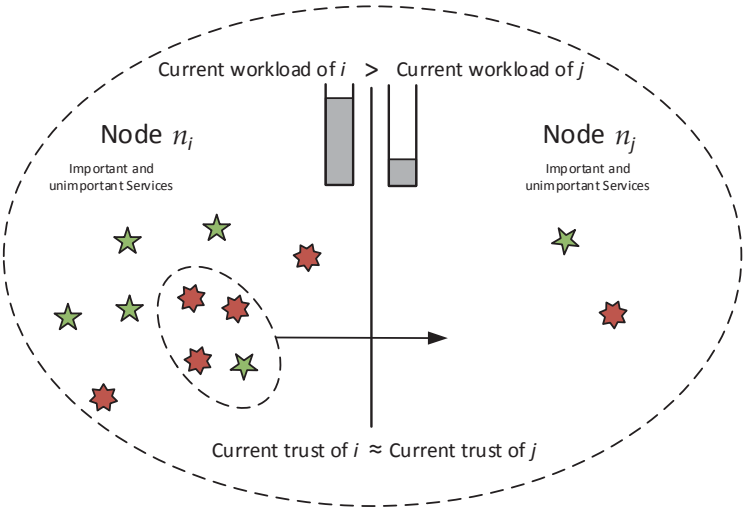


Figure 3.11: The illustration shows how pure load-balancing can be achieved between nodes. Important services are represented with green stars, whereas the unimportant services are depicted with red starlets.

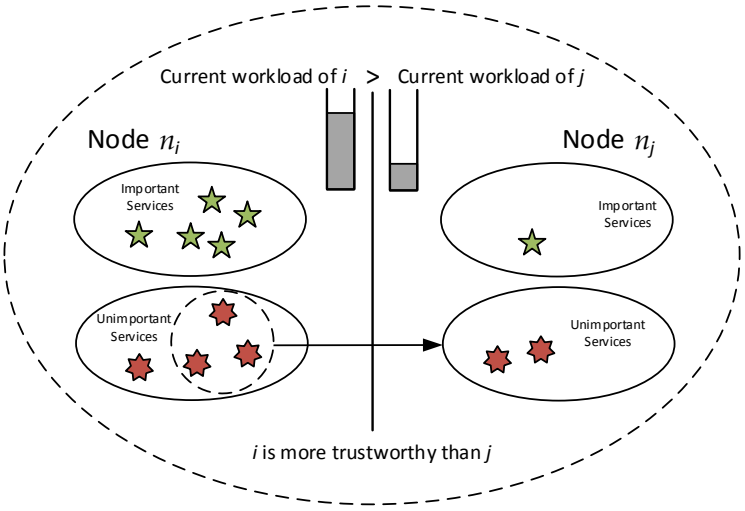


Figure 3.12: A simplified overview of the trust and load optimization strategy. Please note that important services are represented with the green stars, whereas the unimportant services are depicted with red starlets.

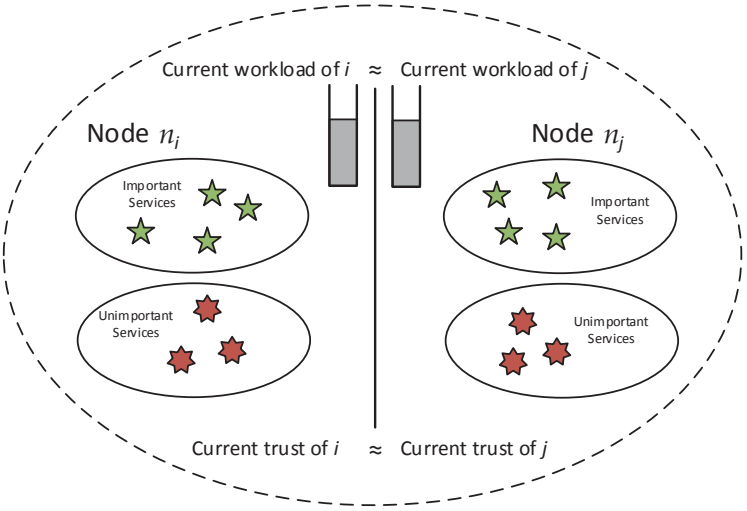


Figure 3.13: Illustration of the non-optimisation strategy used to determine the local termination of the algorithm. Please note that important services are represented with the green stars, whereas the unimportant services are depicted with red starlets.

evaluation results showed that the proposed approach can improve the availability of important services during runtime. However, it makes a small deterioration (i.e., by about 7%) regarding load-balancing. This is due the fact that solutions of this kind represent a trade-off problem in which it is impossible to make any trust distribution better without making at least the load balancing distribution worse. Moreover, the evaluation results showed that the trust-aware self-optimization approach is only for use in situations in which no conflicting trust values between nodes occur. Such conflicts are caused, for example, by collecting trust values independently from the neighbors of a node that can contradict each other. Figure 3.14 visualizes this problem in a short example of three nodes.

Let us consider a network with the three nodes n_1, n_2 and n_3. Let us now suppose that a shielding wall is set between the two nodes, i.e., n_2 and n_3, preventing communication and thus producing poor trust values between them, while the third node n_1 is not affected. In this case, n_2 considers node n_3 as untrustworthy and thus not being able to properly host services. Hence, it wants to relocate important services running on n_3 to another trustworthy node, while contractor n_1 sees no need for action. Such situations cause consistency conflicts during runtime between nodes and must be resolved. Therefore, in Chapter 6 a conflict resolution mechanism is proposed as an extension to the self-optimization algorithm to deal with the trust conflict issue. In the testbed, an average conflict reduction of 97.5%,

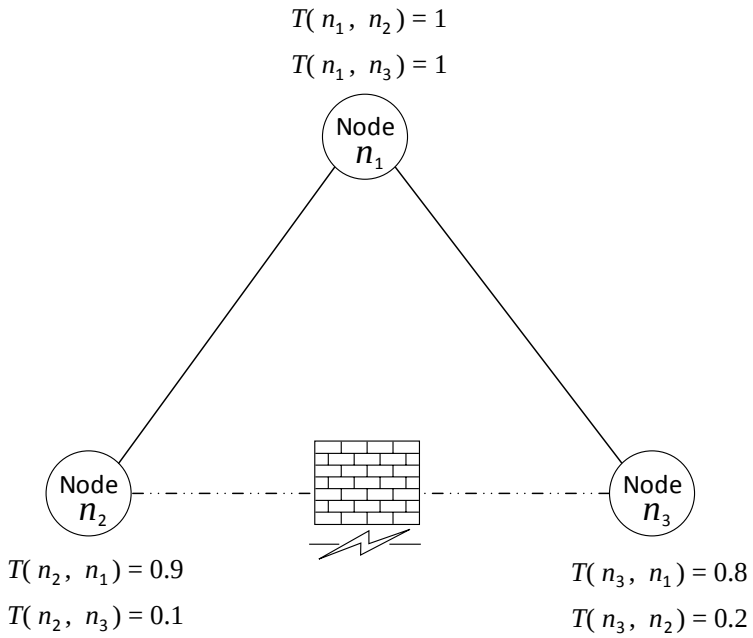


Figure 3.14: The conflicting trust values problem simplified within an example of just three nodes.

53.42% and 6.47% were achieved by the best-case, average-case and worst-case scenarios of the conflict resolution algorithm, respectively.

The Trust-Aware Self-Healing

Self-healing can be defined as the property that enables a system to perceive services that are not operating correctly and, without human intervention, make the necessary adjustments to restore them by using the self-configuration and self-optimization principles (see Chapters 4 and 5). Two ways of thinking have to be considered in the self-healing process, namely *proactive* and *reactive*. The proactive measure enables the system to detect node instability prior to failure which is recognized through degradation of a node's trust value, and then to transfer all running services by using self-optimization techniques to more trustworthy nodes. The strategies used for the service transfer are mainly the same as those described in Section 3.4.3 and thus are not further discussed here. More interesting is the reactive measure. This enables nodes to save recovery information periodically during failure-free execution. Then upon failure, which has to be detected by using a failure detector, a node uses the already saved information to restart from an intermediate state called snapshot, thus reducing the amount of lost computation. In the following, the underlying algorithms of the reactive measure are explained.

Failure detectors. Failure detectors play a crucial role in the development of robust and dependable self-* systems. Assuming that a contractor might crash, the manager has to be able to detect a contractor's failure and take appropriate recovery actions, otherwise the services running on it might block the whole system. Hence, it is important for the manager to regularly monitor its contractors, even if it is a non-trivial task. The main reason for this is the diversity of failures. When a contractor node in asynchronous and distributed environments is not working correctly, it is very difficult for the manager to know the specific cause with certainty: it may be due to a crash failure, execution failure or reachability failure. While there are slight discrepancies in the literature regarding their definitions, in the following the failure models are defined:

1. **Crash failure:** Contractors are considered to execute their services correctly. If a failure occurs at a certain time, the contractor stops permanently. This models a crash of contractor that never recovers by itself. Furthermore, contractors are not able to indicate their failures and stop to send any messages.

2. **Reachability failure:** The contractor is operating correctly but communication channels loose managers' contracts. Consequently, network partitions emerge in the system, which can lead to outdated or duplicated service results if the partitions are merged again into one network.

3. **Execution failure:** The contractor does not halt. It can send messages and answer that it is alive when asked. However, the services which are running on it report wrong results. Services can recover afterwards to the last stored correct state.

- **Regarding (1) and (2):** A well-known technique to cope with crash failures is the *keep-alive* approach [DHS07, PTT12], in the literature also known under the name of the *heartbeat* approach [Sat08]. In this technique, contractors periodically send an alive messages to managers responsible for their monitoring. If, for example, a manager m does not receive such a message from its contractor c after an expiration of time $\Delta_{Timeout}$, it adds c to its list of suspected contractors, as seen in Figure 3.15. If m later receives an alive message from c, then m removes c from its list of suspected contractors. This technique is defined by two parameters:

  – The frequency period Δ_c is the time frequency at which alive messages are sent from c to m.

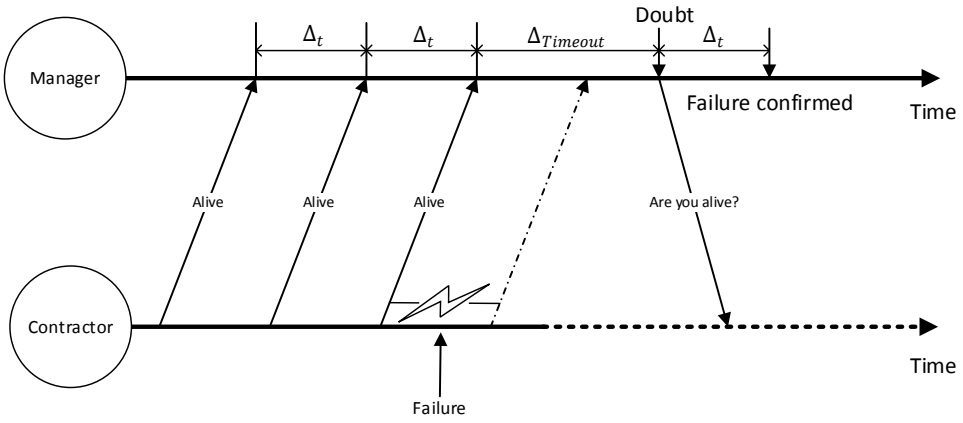


Figure 3.15: The most common approach used in literature to monitor the failure of contractors. This is based on the *pull* model in which contractors regularly unicast a message saying they are alive.

– The timeout delay $\Delta_{Timeout}$ is the time between the last reception of an alive message from c and the time where m starts suspecting c, until an alive message from c is received.

Adjusting Δ_c and $\Delta_{Timeout}$ during runtime makes a trade-off in the system. If these parameters are chosen too short, then failures are detected quickly (i.e., short failure detection delays) but more alive messages are sent in the network. A longer choice of these parameter values results in a larger failure detection delay but less communication overhead. Because of this trade-off problem, it is obvious that an optimal solution minimizing the two criteria simultaneously could never be reached in practice. Our contribution are robust approaches presented in Chapter 7 that provide good but not necessarily optimal solutions to this trade-off problem. They make use of trust concepts to reduce the expected detection delay of failures and their subsequent message cost by allowing more trustworthy contractors to be monitored less frequently than the untrustworthy ones. However, the difference between our approaches arises in the way to determine Δ_c and $\Delta_{Timeout}$ either discrete, continuous or continuous-discrete. The facet of trust considered in this part of the work concerns the availability aspect. For this reason the trust values of contractors are determined based on their uptime in the last interaction steps. Evaluation results showed that the continuous-discrete approach performs best. It can adapt faster to changing trust conditions in the network than the two other approaches and is therefore considered suitable for our TEM middleware. After the occurrence of a failure detected, the next bar-

rier is to determine its type. This can be either a crash failure or reachability failure between the manager and the contractor. Chapter 4.4 introduces the technique we use to differentiate between these two type of failures.

- **Regarding (3):** An important task for the manager is to check whether a contractor is reporting wrong results or not. For masking such execution failures, several approaches are known in literature. One approach which has received much attention in the recent years is the *redundant execution* [AÖK10, EAT13, AVFK02]. This approach enables to detect an execution failure by executing a service at least two times from different contractors and comparing the results. If the results are similar, then the service is run without failure, otherwise an execution failure has happened during the execution. One drawback, however, is that some services cannot be handled by redundant execution, e.g., I/O-based services that might not return the same result in different runs. But as it goes beyond the scope of this work it is not further discussed here.

  Service recovery. As clarified in the introduced failure model, contractors in our system are subject to crash failures. Keeping that fact in mind, a manager has to store the stepwise results of its contractors in a trustworthy place in order to get them back in case of failure. This has the benefit to later reduce the recovery time by restarting the services not from the beginning but rather from an intermediate state. To ensure this, a robust data storage is needed for our system that must obey to the following identified research points:

  – *How to adjust the amount of replicas in the system during runtime in order to guarantee a good availability of service data, characterized for example by five nines availability* $\approx 99.999\%$ *?*

    The answer of this question results, as expected, in a trade-off problem between performance overhead and availability. It is easy to see in Figure 3.16 that the use of a higher number of replicas generally increases the availability of the stored data. However, this availability is only improved until 3 replicas have been reached. A greater number than 3 does not improve the availability any more but makes performance worse, since more replicas imply more resource consumption and higher management cost. Therefore in Chapter 8, we provided a replication approach that enables us to calculate the minimum number of replicas needed for a desired degree of availability taking into account the average availability of nodes. The results of this approach attest a very good reduction of performance overhead in the network. More results

related to the runtime categorization of services based on other factors like checkpoint size and service centrality are, however, left for future work.

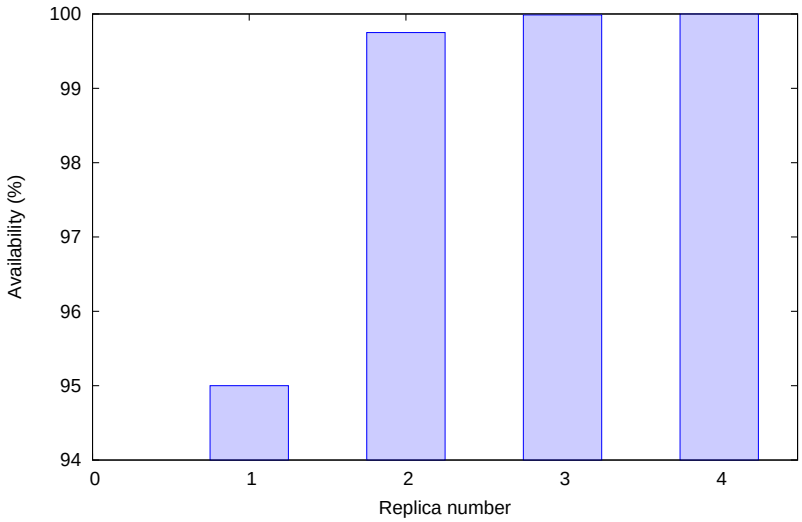


Figure 3.16: Example showing the number of replica that are required for an average nodes' availability of 95%.

– *How to distribute the replicas in a way that the more important replicas are always hosted only on most available nodes, while at the same time achieving load balancing between nodes?*

  This issue can be reduced with minor variations to the same problem that we have addressed in Section 3.4.3: Instead of using the reliability facet, the system focuses on the availability facet by observing the uptime of nodes. The self-configuration algorithm then uses these availability values to perform an initial distribution of replicas on nodes. At runtime, we make use of self-optimization techniques to continuously optimist their assignments. A comprehensive overview of that concept can be found in Chapter 8.

3.5 Application Case Studies

The trust aware self-* properties introduced in this chapter concern basic middleware concepts to provide guarantees of reliability, availability and scalability for the TEM. Apart from these properties, the TEM implements mechanisms that allow the application running on top of it based on the trust metrics to measure uncer-

tainty at runtime and to take the trustworthiness of the applications' entities into account when making decisions. Within the OC-Trust research group several application case studies are investigated for the TEM. The first case study comprises an application from the domain of multi-agent systems called the Trusted Computing Grid. The second application case study, the Autonomous Virtual Power Plants, stems from the domain of decentralized energy management systems. And the last case study is the Trusted Display Grid taken from the domain of multi-user multi-display systems (see Figure 3.17). All these case studies have the main goal to utilize trust at the application level. They are implemented on the TEM to make use of the trust metrics and to profit from the robustness of TEM provided at the middleware level by means of the self-* algorithms described above.

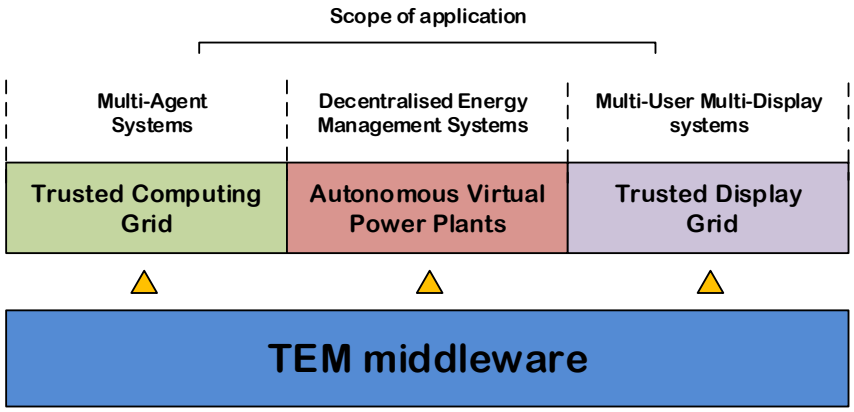


Figure 3.17: The TEM used as basis for the construction of some trustworthy self-organizing applications.

The Trusted Computing Grid. The first application case study that profits from the TEM middleware is the Trusted Computing Grid. This consists of a great number of client computers with different resources that work together in a grid to cooperatively process computationally intensive tasks, e.g., face recognition [Rob09] or ray tracing [Gla89]. Each client in the grid takes on one of two roles related to the execution of an individual task: *submitter* or *worker*. A *submitter* is responsible for breaking down the task into work units, scheduling the execution of work units, and collecting the results of their execution. A *worker* is responsible for the actual execution of work units. However, not every worker in the grid is equally interested to process work units. There are, for example, some workers that might plan to exploit the system by accepting work units and canceling their processing, so called *Egoistic Workers* [ESJ+15]. By making use of trust, the submitters can identify those untrustworthy workers and form Trusted Communities (TCs) [KBA+13].

This represents a community formed by workers and submitters with strong mutual trust relationships that aims to reduce the probability of receiving invalid results. Each Trusted Community is managed by an elected trustworthy manager, called the *Trusted Community Manager* (TCM), which has as goal to maintain the stability of community members using self-* properties. This TCM is an example for an important service in the TEM middleware, since its failure can deteriorate the entire TC.

The Autonomous Virtual Power Plants. Another example of application that profits from the TEM middleware is the Autonomous Virtual Power Plants presented in [SASR13]. This is a power management system composed of a variety of power plants which provide either *dispatchable* or *non-dispatchable* power production. The dispatchable production is made by power plants whose output can be determined in advance such as coal or atomic power plants. In contrast, the non-dispatchable production is made by power plants whose output is unpredictable like wind turbines or solar power plants. One of the main challenges posed here is to maintain the balance between power production and consumption at all times. This is a non-trivial task for non-dispatchable power plants since their production depends on the availability of natural resources like sunlight, air, or wind, that cannot be controlled by humans. The Autonomous Virtual Power Plants application overcomes this issue by integrating trust to allow an automatic regulation of non-dispatchable power plants, so that the dispatchable power plants can be used only as needed [ASS+15, KASR15]. This allows the formation of Autonomous Virtual Power Plants (AVPPs), that group dispatchable and non-dispatchable power plants together based on the Observer/Controller architectural pattern [SENR13]. If an observer identifies during runtime that the organisational structure of an AVPP is not suited, e.g., because maybe one or more AVPPs cannot maintain the power balance any more, the controller performs a new organization of AVPPs in order to bring back the power balance optimized in the system again [ASR15]. The autonomous organization of AVPPs is an essential aspect of this application case study and is therefore considered as an example for an important service in the TEM middleware.

The Trusted Display Grid. The third application case study that profits from the TEM middleware is the Trusted Display Grid [WHKA14]. This allows users to interact in the system with multiple self-organizing displays at the same time. The displays are divided into two types of usage: *private* and *public*. Private displays are those that can protect the personal data of users from external observation like smart phones and tablets. In contrast, public displays are those that everyone can

use, whose data is public and can be shown in presence of other people like Microsoft Surfaces [3]. The major challenge here is to self-organize the transfer of data between public and private displays at runtime. However, such an organization is a non-trivial task. This is because the displays should continuously protect the users' privacy on the one hand and on the other hand maintain the user's acceptance. Otherwise, the user might abandon the system. The Trusted Display Grid tackles this problematic situation by using the User Trust Model (UTM) [HWA15], which allows the displays to monitor the interaction of users, to measure their current trustworthiness, and to apply appropriate self-organizing mechanisms to increase the users' trust as well as usability in the system. This UTM is essential for the operability of the Trusted Display Grid and is therefore treated as an important service by the TEM middleware.

3.6 Conclusions and Future Work

In this chapter, a middleware architecture for trustworthy self-* properties in open distributed systems is presented. The design of the architecture relies on trustworthy algorithms to provide guarantees of reliability, availability and scalability to service-oriented middlewares. The baseline middleware used in this work is OCμ. We have explained how the trustworthy self-* layer can be easily integrated into OCμ to make it more robust in the face of untrustworthy components. The resulting middleware is TEM, a trust-enabling middleware that can profit from the advantage of trust and OC principles at the same time. The TEM makes use of different trust metrics, i.e., such as direct trust, reputation, and confidence to monitor the behavior of nodes in the system at runtime. This monitoring is very important to guide and control the self-organization process between nodes by means of trust-aware self-* properties. The self-* properties examined in this work are self-configuration, self-optimization, and self-healing. We believe that these properties are fundamental for the design of every autonomous, scalable and fault-tolerant service-oriented middleware. The self-configuration is related to the ability to perform an initial distribution of services on nodes taking the resource requirement and importance level of services into account. The self-optimization focuses on optimizing the allocation of services at runtime by monitoring the trust and resource consumption of nodes. And the self-healing aspect is concerned with the ability to handle failures of nodes in order to guarantee that all services running on them stay available even in case of network partitions. We applied the TEM middle-

[3] Microsoft Surface – [Accessed: October 21, 2015] – `http://www.microsoft.com/en-us/pixelsense/whatissurface.aspx`.

ware to different application case studies and clarified how uncertainty in open environments can be mastered by using trust. Due to the fact that future application services will become more autonomous, we expect to see more self-* systems based on our (or a similar) architecture. The future design of self-* middlewares will increase the demand of trustworthy self-* properties to ensure robustness in the system. The architecture presented in this chapter is a step in this direction.

4

Trust-Aware Self-Configuration

Abstract. This chapter presents a trust enhancement of the self-configuration algorithm based on the well-known Contract Net Protocol. This baseline algorithm can be used in a distributed system, i.e., multi-agent system, cloud computing or grid system, to equally distribute the load of services on the nodes. However, the trust enhancement of self-configuration assigns services with different importance levels to nodes so that more important services are assigned to more reliable nodes. Evaluations have been conducted to rate the effectiveness of the algorithm when nodes are failing, i.e., the reduction of failures of important services. The results show that our self-configuration algorithm increases the availability of important services by more than 12%.

4.1 Introduction

The growing complexity of today's computing systems requires a large amount of administration, which poses a serious challenging task for manual administration. Therefore, new ways have to be found to autonomously manage them. They should be characterized by so-called self-* properties such as self-configuration, self-optimization, and self-healing. The autonomous assignment of services to

nodes in a distributed way is a crucial part for developing self-configuring systems. In this chapter, a trust-aware self-configuration algorithm for self-* systems is presented, which aims on the one hand to equally distribute the load of services on nodes as in a typical load balancing scenario and on the other hand to assign services with different importance levels to nodes so that the more important services are assigned to more trustworthy nodes. The trust definition [SKL+10] adopted for this work is the definition provided by the research unit OC-Trust of the German Research Foundation (DFG) by regarding different facets of trust, as, for example, safety, reliability, credibility and usability. The focus here lies on the reliability aspect. Furthermore, it is assumed that a node can not realistically assess its own trust value because it trusts itself fully. Therefore, the calculation of the trust value in this work must be done with the previously introduced trust metrics presented in chapter 3.4. Recall, the value t_n represents the current trust of node n and will always range between 0 and 1. The value of 0 means that n is not trustworthy at all while a value of 1 stands for complete trust. With trust information, nodes of a system have a clue about which nodes to cooperate with, and this is important for self-configuring systems. The chapter offers as contribution the following aspects:

(i) a decentralized self-configuration algorithm for the initial allocation of services on nodes taking into account trust to increase the availability of important services in open distributed environments (see Section 4.3),

(ii) some coverage metrics for the calculation of the quality of service in order to determine the suitability of nodes to host services (see Section 4.3.1),

(iii) in the case of conflicting assignments a conflict resolution mechanism is used to solve the conflict without any further message (see Section 4.3.3),

(iv) a fault handling approach to allow a manager to monitor its contractors after the distribution of services (see Section 4.4), and

(v) a set of several coordination strategies enabling the system to operate with multiple managers at the same time and to carry out several distribution phases simultaneously (see Section 4.5)

All aspects are evaluated and discussed with respect to a toolkit based on the TEM [ASM+13], a trust-enabling middleware for building real-world distributed Organic Computing systems. The main results of our self-configuration algorithm are provided in Section 4.3.4. Section 4.5.1 demonstrates benefits of the proposed coordination strategies to support parallelism of multiple managers at runtime. Finally, the chapter is closed with a conclusion and future work in Section 4.6.

4.2 Related Work

There are many sophisticated approaches to deal with the allocation problem of services on nodes, either to achieve good load balancing or to minimize energy consumption.

An approach that has become a standard by FIPA[1] is the Contract Net Protocol [Smi80]. It consists of finding an agent that is the most suitable to provide a service. This approach is often adapted and applied in many application domains, for example, manufacturing systems [HC09], resource allocation in grids and sensor web environments [KB09] [GG08], as well as in hospitals [DGB03], electronic marketplaces [DKRA00], power distribution network restoration [KHS+09], etc. Our model is based on the Contract Net Protocol, extended by trust. In this context, trust serves as a mean to give nodes a clue about with which nodes to cooperate.

Bittencourt et al. [BMCB05] presented an approach to schedule processes composed of dependent services onto a grid. This approach is implemented in the Xavantes grid middleware and arranges the services in groups. It has the drawback of a central service distribution instance and therefore a single point of failure can occur.

A biologically inspired solution for load balancing in distributed environments called Honeybee Foraging is presented in [RTBL08, RLTB10]. This approach works based on the behavioral motion of honeybees who fellow each other in harvesting food. The bees returning from hive provide information to other bees about the status of resting food in the hive. In the same way, resource nodes are allocated to incoming services based upon information provided by previous services regarding the global load in the whole system. In contract to our approach, this technique does not take the priority of different service classes into account.

Trumler et al. [TKU06] described a scheduling algorithm for distributing services onto nodes based on social behavior. It is implemented in the OCμ middleware. In their model, nodes can calculate a QoS for the services to decide which service is assigned to which node. In this case only resource constraints are used to describe cases when a service should be hosted depending on a specific hardware. In contrast to our approach, this algorithm does not include trust constraints.

In [THW02], Topcuoglu et al. presented an approach to consider the priorities of tasks. They try to select tasks in order of their priorities and to schedule them to the best machine that minimize their finish time in an insertion based manner. This approach has been shown to significantly improve the schedule computation time.

[1] FIPA: Interaction Protocol Specifications - [Accessed: November 6, 2015] `http://www.fipa.org/specs/fipa00029/`

However, a disadvantage is that important tasks might run on unreliable nodes and are prone to fail.

Later, in [BEDL13], reliability constraints were considered to find a homogeneous allocation of the instances of services. Contrary to this work, our approach is able to work with heterogeneous systems. More precisely, we are looking for a scalable allocation approach that has neither a central control nor complete knowledge about the system with the property to solve the initial trust and load allocation problem of heterogeneous services solely based on reliability measurements, greatly improving the availability of important services in open distributed environments. Furthermore, the algorithm must includes a fault handling mechanism enabling the system to continue hosting services even in the presence of faults. Parts of the content of this chapter have been published by the author in the following workshop, conference and journal:

- **[MKFU14]:** Nizar Msadek, Rolf Kiefhaber, Bernhard Fechner, and Theo Ungerer. *Trust-Enhanced Self-Configuration for Organic Computing Systems*. In ARCS 2014: Proceedings of the 27th International Conference on Architecture of Computing Systems, pages 37-48, Lübeck, Germany, Springer, 2014.
- **[MKU14a]:** Nizar Msadek, Rolf Kiefhaber, and Theo Ungerer. *Simultaneous Self-Configuration with Multiple Managers for Organic Computing Systems*. In SAOS 2014: Proceedings of the second International Workshop on Self-Optimisation in Organic and Autonomic Computing Systems in conjunction with ARCS 2014, pages 1-7, Lübeck, Germany, IEEE Computer Society, 2014.
- **[MKU15a]:** Nizar Msadek, Rolf Kiefhaber, and Theo Ungerer *A Trustworthy, Fault-tolerant and Scalable Self-Configuration Algorithm for Organic Computing Systems*. In JSA 2015: Journal of Systems Architecture, issn 1383-761, `http://www.sciencedirect.com/science/article/pii/S138376211500082X`

4.3 The Trust-Enhanced Self-Configuration

This section presents a trust enhancement of the baseline self-configuration algorithm based on the well-known Contract Net Protocol [Smi80]. This baseline algorithm can be used in a distributed system, i.e., multi-agent system, cloud computing or grid system, to equally distribute the load of services on the nodes. However, the trust enhancement presented in this section aims mainly on finding a robust distribution of services by including trust. The services are categorized into important services with a high required trust, and non important services with a low required

trust. Important services are those, which are necessary for the functionality of the entire system. E.g., Bernard et al. [BKHC10] present a computing grid to solve computationally intensive problems. In their model, trust is incorporated to enable nodes to form Trusted Communities (TCs). The manager, that administrates these TCs is an example for an important service, since its failure deteriorates the entire TC.

The goal is to maximize the availability of important services. Therefore, it is necessary to assign important services to more trustworthy nodes. Trust in this context is expressed by a trust value based on previously introduced trust metrics presented in chapter 3.4. In addition to trust, resource requirements (e.g., like CPU and memory) should also be considered to balance the load of the nodes.

4.3.1 Metrics

The self-configuration focuses on assigning services with different required trust levels to nodes which have different trust levels so that more important services are assigned to more trustworthy nodes. Furthermore, the overall utilization of resources in the network should be well-balanced. Therefore, a metric is defined in 4.1 to calculate a Quality of Service (QoS_{total}), i.e., the suitability of node to host a specific service.

$$QoS_{total} = (1 - \alpha) \cdot QoS_{trust} + \alpha \cdot QoS_{workload}. \tag{4.1}$$

The relationship between trust and workload can be set through $\alpha \in [0, 1]$. If $\alpha = 1$, the QoS_{total} is only obtained by the current value $QoS_{workload}$, i.e., the suitability of a node to host a specific service with regard to its workload. If $\alpha = 0$, the QoS_{total} is decided only by the actual QoS_{trust} value, i.e., the suitability of a node to host a specific service with regard to its trust value. A higher value α favors $QoS_{workload}$ over QoS_{trust}.

- **QoS_{trust}** indicates how well the trustworthiness of a node fulfilled the required trust of a service. Figure 4.1 visualizes formula 4.2 to calculate the QoS_{trust}.

  t_n represents the current trust of node n calculated based on previously introduced trust metrics presented in chapter 3.4 and will always range between 0 and 1. The value of 0 means that n is not trustworthy at all while a value of 1 stands for complete trust. In this work, it is assumed that t_n is constant at a certain point in time. However, t_n is likely to change over time. This issue will be addressed in depth in Chapter 5. The value t_s represents the required trust value of service s defined by the user according to the importance level of the

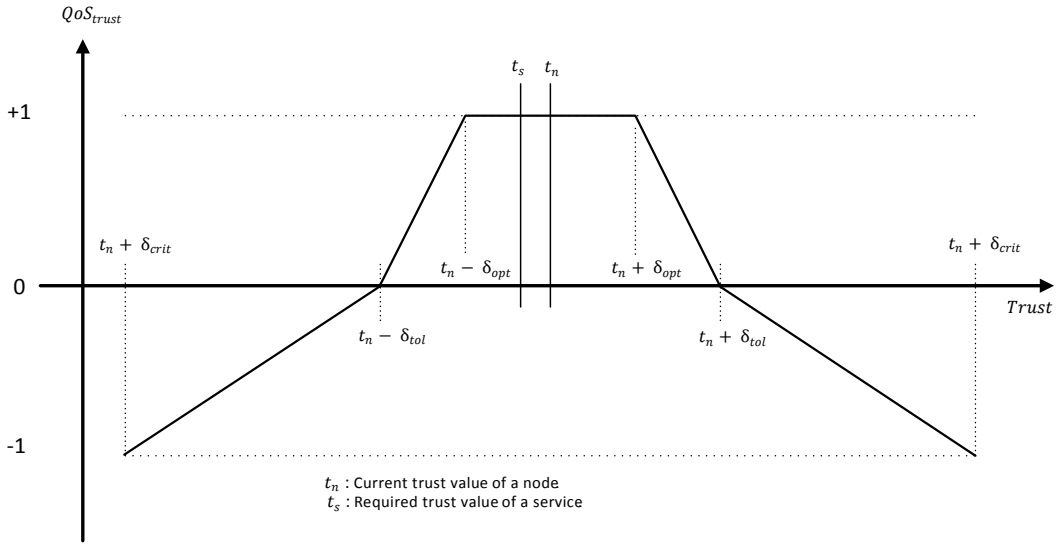


Figure 4.1: QoS_{trust} based on the difference between the trust t_n of node n compared to the required trust t_s of service s

service. The value of 1 means that the service s is very important and requires to be hosted only on trustworthy node while a value of 0 stands for an unimportant service and means that s can tolerate to be hosted on untrustworthy node. If both values are close enough then n has fulfilled the required trust value of a service s. Close enough is defined by the threshold δ_{opt} (optimal area). If the difference between t_n and t_s is more than δ_{opt}, then QoS_{trust} will be gradually decreased until it reaches 0 at $t_n \pm \delta_{tol}$ (tolerance area). If t_s is even beyond $t_n \pm \delta_{tol}$ then the QoS_{trust} will drop below 0 (critical area). In the case that the divergence between t_n and t_s is more than δ_{crit}, the QoS_{trust} remains constant at -1. This is expressed by the formula 4.2, with $t_s, t_n \in [0, 1]$ and $0 < \delta_{opt} < \delta_{tol} < \delta_{crit}$. Please note that our interest for the metric introduced here focuses only on a symmetric behavior of QoS_{trust} in order to ensure a fair allocation of services on nodes. This means that a lower QoS_{trust} value is given not only for important services trying to be hosted on untrustworthy nodes, but also for unimportant services trying to be assigned on trustworthy nodes. Otherwise, the important services might be handicapped later in the distribution phase (see Section 4.3.2) to find still unloaded trustworthy nodes which can be hosted on.

- $\mathbf{QoS_{workload}}$ gives an estimation of the workload of a node. As long as the load of a node is lower than two times its maximum capacity, the quality of service (QoS) decreases linearly, otherwise it remains constant at -1. It

$$QoS_{trust}(t_s) = \begin{cases} 1 & \text{if } 0 \leq |t_n - t_s| \leq \delta_{opt} \\ \frac{-|t_s - t_n| + \delta_{tol}}{\delta_{tol} - \delta_{opt}} & \text{if } \delta_{opt} < |t_n - t_s| \leq \delta_{tol} \\ \frac{-|t_s - t_n| + \delta_{tol}}{\delta_{crit} - \delta_{tol}} & \text{if } \delta_{tol} < |t_n - t_s| \leq \delta_{crit} \\ -1 & \text{otherwise} \end{cases} \tag{4.2}$$

is assumed that the capacity of a node has not a hard limit, e.g., swapping data from RAM to hard drive provides extra memory at the cost of runtime. Please note that swapping has an extremely high cost and will noticeably degrade the overall performance of the host node. Therefore in this work, it is assumed that only swapping with two times of the nodes's capacity is tolerated and not three times or more. The QoS regarding only one resource i is calculated using Formula 4.3 as follows:

$$workload_i(V_{req_i}, V_{av_i}) = \begin{cases} \frac{V_{av_i} - V_{req_i}}{V_{max_i}} & \text{if } V_{req_i} \leq V_{av_i} + V_{max_i} \\ -1 & \text{otherwise} \end{cases} \tag{4.3}$$

$$QoS_{workload} = \frac{1}{n} \sum_{i=1}^{n} workload_i(V_{req_i}, V_{av_i}) \tag{4.4}$$

with $V_{max_i} > 0$, $V_{req_i} > 0$, and $V_{av_i} \leq V_{max_i}$. It is to note that V_{req_i} means the required resource i of a service. The available resource amount of a node is denoted by V_{av_i} and its maximum resource amount by V_{max_i}. However, every node can have multiple resources n. Therefore the $QoS_{workload}$ is calculated by the average sum of all resource values (See Equation 4.4).

4.3.2 Self-Configuration Process

This section discusses the methodology for distributing services. This consists of a collection of services with different importance levels which should run on nodes with different trust values. It is known to be a NP-hard problem to find an optimal solution for the distribution of the services on the nodes, so that the quality of service is optimal [Rei99]. Furthermore, there is no known algorithm which can, for a given solution, in a polynomial time identify whether it is optimal. The aim behind self-configuration is to find a distributed and robust but not necessarily optimal yet good enough solution.

The quality of service metric presented in Section 4.3.1 is used to evaluate the distribution phase which is based on the Contract Net Protocol [Smi80]. During the distribution phase, every node in the network can act as a manager or contractor. A

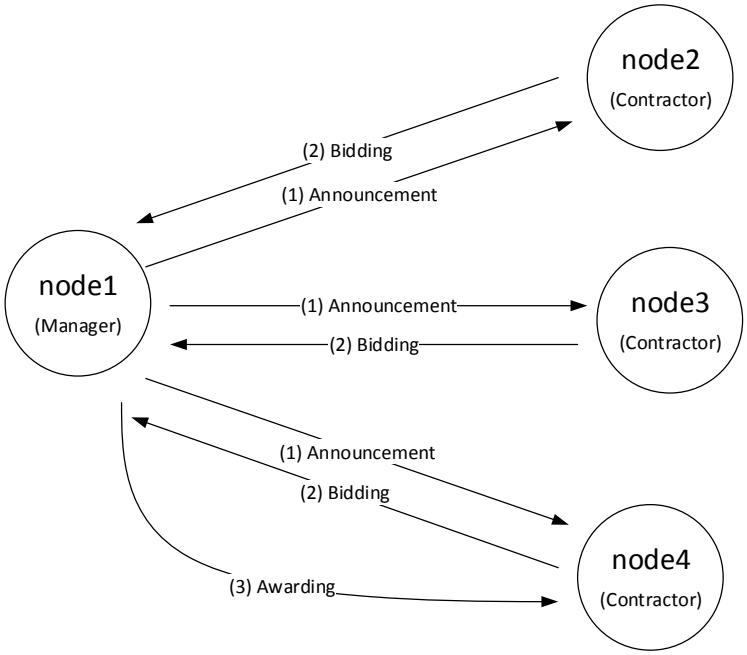


Figure 4.2: Elementary representation of the distribution phase

manager is responsible for assigning services. A *contractor* is responsible for the actual execution of the service. An algorithm that helps contractors to autonomously elect their own managers can be found in [MSKU15]. For the sake of simplicity, in the self-configuration process, the usage of multiple managers is omitted. Figure 4.2 visualizes a step-by-step example on how the negotiation process is run between nodes.

1. **Announcement:** The manager (e.g., node1) that wants to distribute a service, initiates contract negotiation by advertising the existence of that service to the other contractors (e.g., node2, node3 and node4) with a service announcement message. A service announcement can be transmitted to a single contractor in the network (unicast), to a specific set of contractors (multicast) or to all contractors (broadcast).

2. **Bidding:** Every contractor node that receives the announcement calculates the value $QoS_{workload}$ for the given service based on its own available resources and then submits its bid in form of $QoS_{workload}$ to the manager. Note that the service announcement is ignored if the service cannot be hosted due to missing resources.

3. **Awarding:** When the expiration time (i.e., the deadline for receiving bids) has passed, the manager that sent the service announcement must calculate

QoS_{trust} for every contractor in order to build the QoS_{total} and decide whom to award the contract to. In the basic Contract Net Protocol the manager selects among the received bids the contractor with the highest $QoS_{workload}$. Our enhancement improves the Awarding phase by including trust (QoS_{trust}) so that more trustworthy contractors always have a higher chance to receive the service than less trustworthy contractors. The result of this process will be then communicated to the contractors that submitted a bid. It is to note that the expiration time is defined by the user.

4.3.3 Conflict Resolution

During the self-configuration process, several nodes could be ranked with the same QoS_{total}. This might lead to a conflict for the manager to decide to whom to award the service. To avoid this a conflict resolution mechanism is used which does not need any further messages. The conflict resolution mechanism consists of three stages which might be used in the following chronological order:

1. **Minimum latency:** The node with the lowest latency will get the service.
2. **Minimum amount of already assigned services:** The node with the least amount of already assigned services will get the service, assuming that a lower amount of services will produce less load (e.g., a thread switch or process switch would produce additional load).
3. **Random assignment:** It is unlikely but not impossible that all of the former values were equal. In this case one node will be selected at random to get the service.

4.3.4 Evaluation

In this section an evaluation for the introduced self-configuration approach is provided. For the purpose of evaluating and testing, an evaluator tool based on our TEM [ASM+13] middleware has been implemented which is able to simulate the distributed self-configuration process. The system network consists of 50 nodes, where all nodes are able to communicate with each other using message passing. Experiments with more nodes were tested, and yielded similar results, but with 50 nodes more observable effects for the two first parts of the evaluation were seen. Each node has a limited resource capacity (e.g., CPU and memory) and is judged by an individual trust value without any central knowledge. Notice that the trust values will always range between 0 and 1. The value of 0 means that the node is

Node Type	CPU (MHz)	Memory (MB)	Trust	Amount (%)
Type 1	200-800	500-1000	0.7-0.9	10
Type 2	500-1500	500-1500	0.3-0.6	50
Type 3	1500-2000	2000-4000	0.4-0.8	30
Type 4	2000-3000	4000-8000	0.4-0.9	10

Table 4.1: Mixture of heterogeneous nodes

not trustworthy at all while a value of 1 stands for whole trust. Four type of nodes are defined with different resources and trust levels (see Table 4.1).

Then 150 services, 75 of them important and 75 unimportant (50/50 ratio), with random resources (CPU $\in [0, 800]$ and RAM $\in [0, 1000]$ are generated so that all nodes are loaded on average to 60%. Without additional information important services might run on untrustworthy nodes and are prone to fail. Such situations can be avoided. Using trust as a constraint, the trust of a node can be measured and taken into consideration for the service distribution. Hence, the goal is to maximize the availability of important services. Therefore, it is necessary to assign the more important services to more trustworthy nodes. In the following the results of the conducted evaluations are presented.

Quality of Distribution. To evaluate the distribution of important services with regard to trust, the mapping between the trust of the node and the required trust of the service is compared using different values for α. If $\alpha = 1$, the service distribution is only obtained by considering the resource utilization as in a typical load balancing scenario. Figure 4.3 shows the results of this experiment with $\alpha = 1$, whereas the values on the x-axis represent important services together with nodes and their trust is depicted on the y-axis. The dotted line represents the expected trust of important services sorted in descending order. However, the rectangular points show the trust of nodes on which an important service is running. In the majority of cases, the divergence between both values, i.e., the current trust of a node and the required trust of the service is very important. This explains why the majority of important services are hosted on untrustworthy nodes.

To overcome this issue, trust has been taken into consideration. Figure 4.4 illustrates exactly the same information as Figure 4.3, but with $\alpha = 1/2$ to provide a better assigning of important services on trustworthy nodes. Please note that the allocation of services is referred to as the trust and workload trade-off problem in which it is impossible to make any trust distribution better without making at least the load balancing distribution worse. This trade-off depends on the specific as-

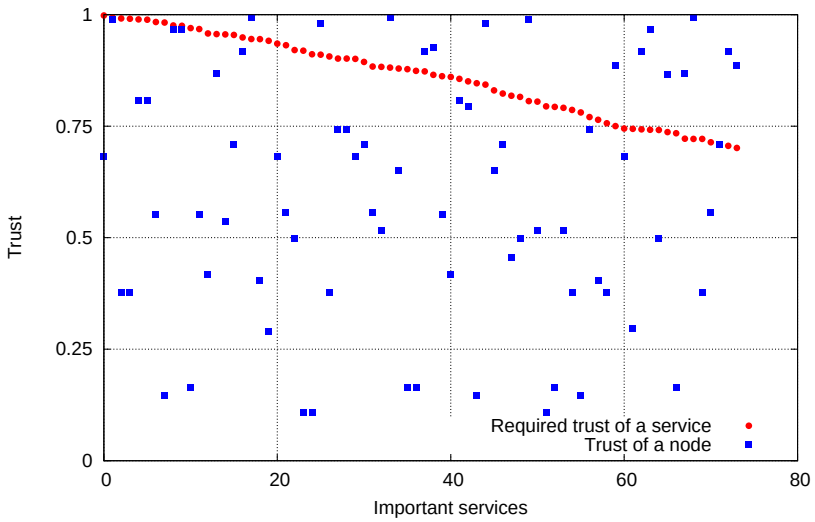


Figure 4.3: Quality of distribution without trust ($\alpha = 1$, i.e, nodes could be shown multiple times in this figure, since they can host multiple services at the same time)

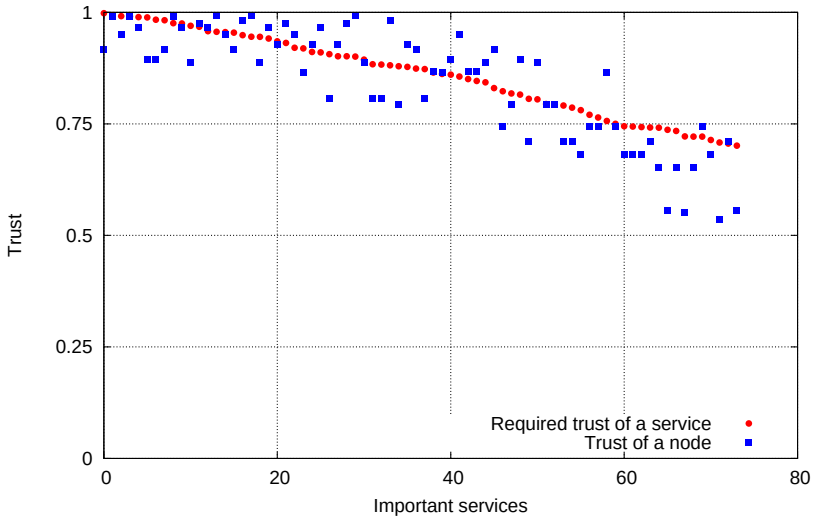


Figure 4.4: Quality of distribution with trust ($\alpha = 1/2$, i.e., nodes could be shown multiple times in this figure, since they can host multiple services at the same time)

sortment of α. Learning the optimal trade-off α between trust and workload could further improve the performance of the proposed self-configuration algorithm and could be a starting point for future work.

Permanent Node Failures. The improved self-configuration algorithm should assure beside load balancing that the majority of important services runs on trustworthy nodes. This can be shown by letting untrustworthy nodes fail and comparing the amount of unavailable important services of the baseline algorithm (i.e., load balancing distribution using Contract Net Protocol) with the trust-enhanced version. However, it should be noted that in a real life situation it is unlikely that all of the untrustworthy nodes fail at once. For this purpose a more realistic approach is adopted using a selection metric to decide which node fails at each time step. This selection metric is based on a roulette wheel selection, where nodes with lower trust values have a higher chance to fail than other nodes with a higher trust values. Our goal is to evaluate the cumulative amount of unavailable important services for the trust-enhanced and baseline algorithm.

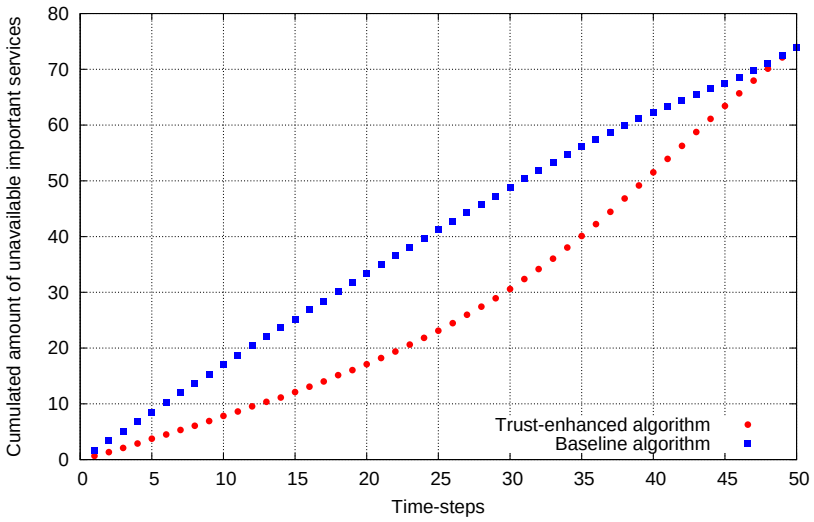


Figure 4.5: Failure of important services

Figure 4.5 shows the results of this experiment. The values on the x-axis stand for time-steps and their cumulative amount of unavailable important services are depicted on the y-axis. Within one time-step, one node is failed based on the selection made by the roulette wheel selection. The dotted line represents the cumulative number of important services that failed per time step using the trust-enhanced algorithm. However, the square line illustrates exactly the same information but with the baseline algorithm. At the first step, the two curves look the same, because all nodes are still running. The most interesting part in the figure is the middle part,

where a number of nodes has failed and the number of still running important services is higher for the trust-enhanced algorithm than in the baseline. At the time step 50, i.e., in that case, all the nodes have failed, all important services are down (exactly 75 services), which explains the similarity again of the two curves. To quantify the obtained enhancement of our approach, the availability enhancement (AE) of important services is calculated as follows:

$$AE = \frac{1}{ts} \sum_{i=1}^{ts} (c_i - t_i) \tag{4.5}$$

where ts means the number of time steps. c_i is the amount of unavailable services at time step i calculated with the baseline algorithm. However, t_i is the amount of unavailable services at time step i calculated with the trust-enhanced algorithm. As a conclusion, we can state that the use of trust leads to positive impact on the QoS metric because an availability enhancement (AE) of more than 12% was achieved with $\alpha = \frac{1}{2}$. In addition, we did not observe any simulation run[2] with the trust metric that showed a bad mapping between trust of nodes and required trust of services.

4.4 Fault Handling Mechanism

As has been indicated in Section 4.3.2, contractors play an important role in the establishment of the self-configuration process. Assuming that a contractor might crash, the manager has to be able to detect contractor's failure and take appropriate self-healing actions, otherwise the services running on it might block forever. Hence, it is important for the manager to regular monitor its contractors. During the self-configuration process two monitoring strategies are supported: *polling* and *pushing*. In the polling strategy as mentioned in Section 3.4.3, the manager periodically sends a ping message to a contractor. If the ping fails, then the manager assumes either contractor has crashed or the reachability problem has occurred. However, in the pushing strategy, a contractor periodically sends an alive message to the manager. If the manager does not receive an alive message within a certain duration then it considers that the contractor hangs. Please note that the pushing strategy is capable of detecting both crash and hang failures, whereas the polling strategy is only capable of crash failures. The user decides based on its need which one of these strategies should be used. Figure 4.6 shows how the monitoring process between a manager and a contractor works. The contractor periodically

[2]In total, about 10000 runs were evaluated.

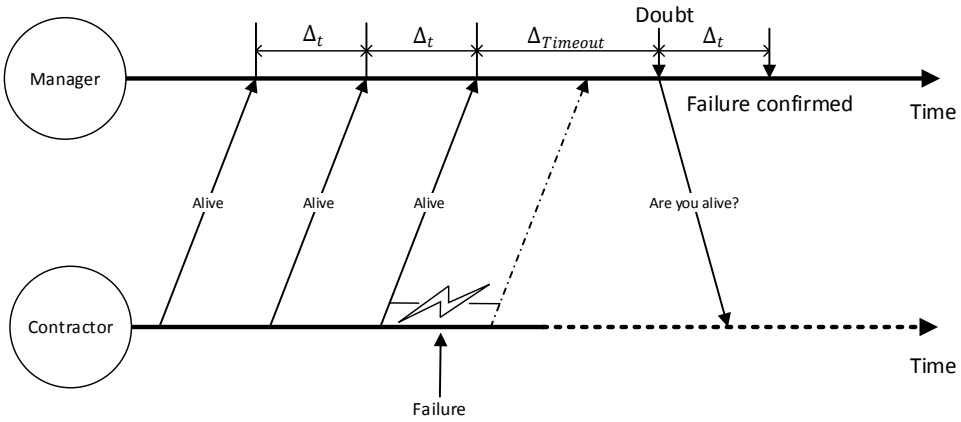


Figure 4.6: Contractor monitoring using the pushing strategy presented in Section 3.4.3.

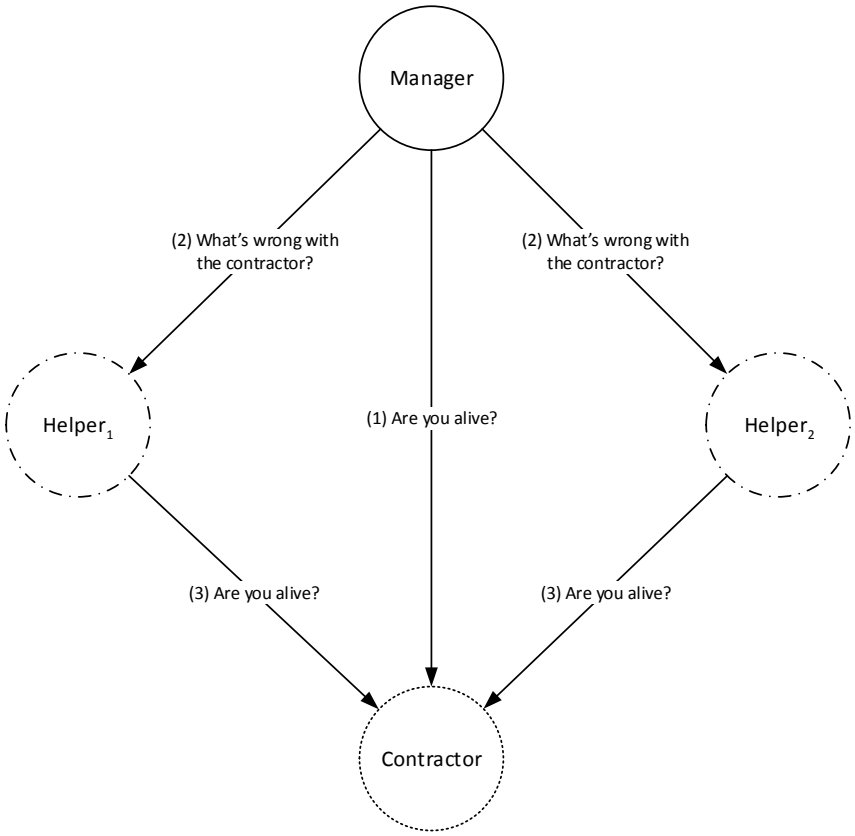


Figure 4.7: Simplified illustration of the detection model showing how to distinguish between crash and reachability failure by making use of helpers. Solid circle represents the manager, dashed circle represents the helpers, and dotted circle represents the contractor. Please note that in the practice, the manager would expect to have more helpers than only the two ones depicted here.

sends to the manager an alive message. When the manager fails to receive such a message from the contractor and after it has waited for a time period of $\Delta_{Timeout}$, it considers that a failure has occurred. Now, it remains to the manager to discover the type of failure. This can be either a crash failure or reachability failure between the manager and the contractor. In order to determine which one of these failures has occurred, the manager needs at least the help of two other nodes, i.e., *helper* 1 and *helper* 2. As shown in Figure 4.7, both helpers send to the contractor, the "*Are you alive?*" messages. If no response is received by both helpers (i.e., *helper1* and *helper2*) from the contractor within a configurable time period, then a crash failure is confirmed. If one of the two helpers, receives the response from the contractor, then a reachability failure is confirmed.

4.5 The Simultaneous Self-Configuration

In this Section, coordination strategies as enhancement are incorporated to the self-configuration process in order to make it scalable. As has been indicated in Section 4.3.2 (and as is shown in Figure 4.2), a manager locates viable contractors via a process of bidding which happens in three stages:

- a manager announces a service,
- contractors evaluate the service with respect to their workload and send bids to the manager.
- the manager combines the received bids with trust, rates them and chooses a contractor to award it the contract.

The self-configuration introduced so far only considers the sequentially assigning of services. This means that it is not suited to operate with multiple managers at the same time. Hence, the basic algorithm neither detects nor resolves conflicts, which is the principal reason why coordination is needed. The simultaneous self-configuration proposes enhancement coordination strategies, which allow multiple managers to carry out several distribution phases simultaneously. In this enhancement, available contractors evaluate service announcements made by their managers and submit bids on those for which they are suited. All these contractors are not completely sure to be contracted when they submit their bids. This means that when a contractor receives a new service announcement it does not know if the earlier announced services are to be awarded at the same time. Figure 4.8 visualizes this concept using a simple example of two managers.

Let us assume two managers, m1 and m2: m1 is responsible for assigning service s1, m2 is responsible for service s2. It is assumed that both services s1 and s2 are

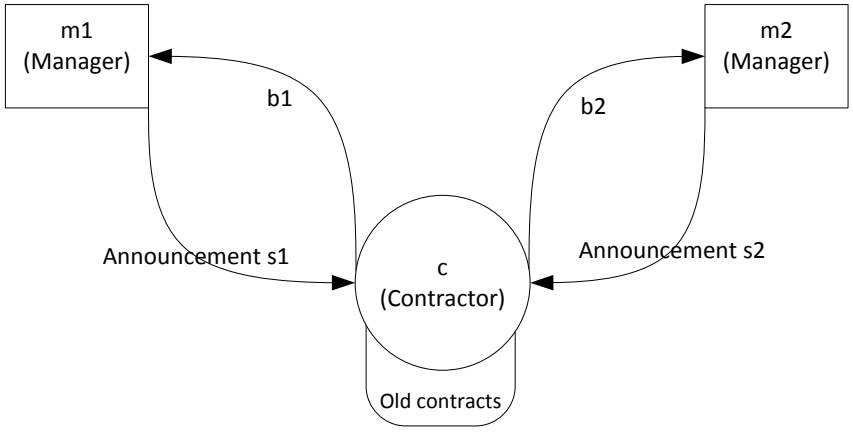


Figure 4.8: Multiple concurrent bids

different and are executed concurrently. Contractor c submits two different bids b1 and b2 to m1 and m2 (i.e., even in the case of a potential overload if both contracts are awarded).

The following presents all possible situations that can happen after a:

- **Non-Coordination:** Contractor c is not awarded with a new contract which is a trivial case, nothing has happened.
- **First Coordination Situation:** Manager m1 sends an award message to c informing him to be the most appropriate. Contractor c sends then an acknowl-

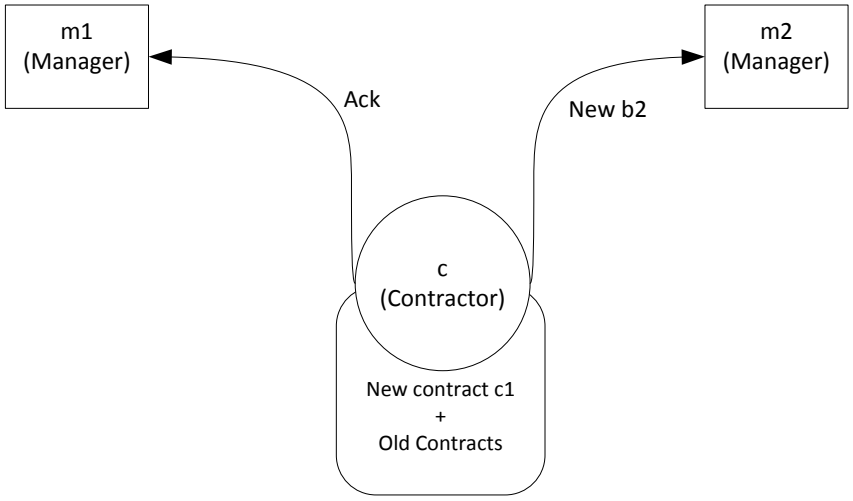


Figure 4.9: Inconsistency resolution

edgment message of the award back to m1 announcing his willingness to perform service s1. This leads to an inconsistent configuration because b2 is now becoming obsolete for manager m2. To avoid this inconsistency, c sends

at the same time a new bid message b2 to manager m2 announcing him the recalculated value of $QoS_{workload}$ for service s2, as shown in Figure 4.9.

- **Second Coordination Situation:** Contractor c receives at the same time a contract c1 from m1 and a contract c2 from m2 leading to an inconsistent and potential overload situation, as shown in Figure 4.10. In order to solve this,

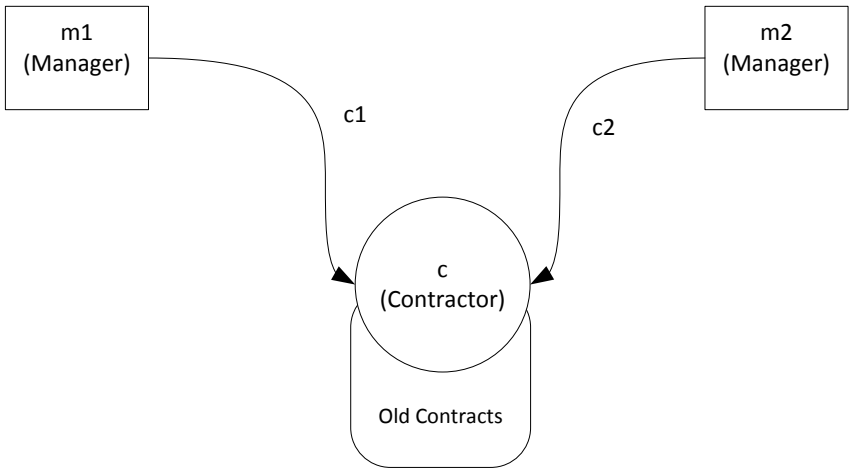


Figure 4.10: Simultaneous contracts

contractor c sends an acknowledgment message to m1 and a new bid message b2 to manager m2, as shown in Figure 4.11. Here the principle of first-come first-serve is adopted. Then, the contract of m2 will be refused.

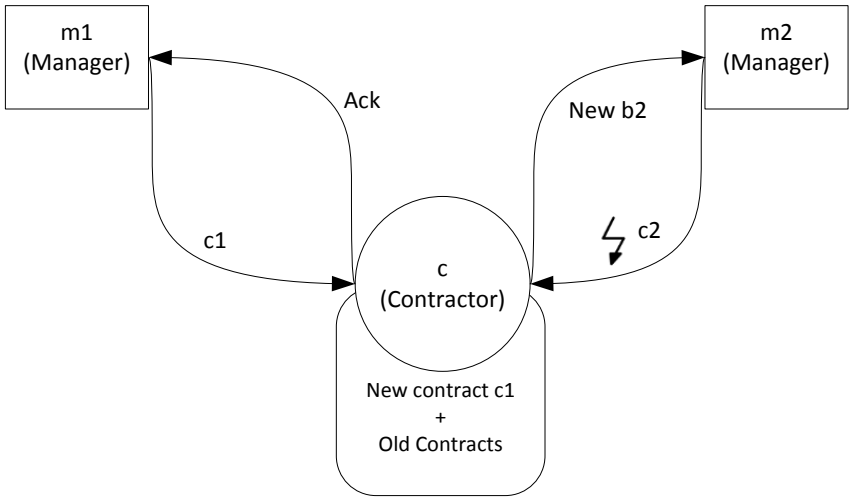


Figure 4.11: Inconsistency resolution

4.5.1 Evaluation

In this section an evaluation for the improved self-configuration approach is presented. For the purpose of evaluating and testing, the basic evaluator has been extended to simulate the simultaneous self-configuration algorithm. The evaluation network consists of 50 nodes, where all nodes are able to communicate with each other using message passing. Each node has a limited resource capacity (e.g., CPU and memory) and is judged by an individual trust value (reliability) without any central knowledge. To keep consistency with the previous experiments, we use the same setting as in Table 4.1. This means that four type of nodes are also defined with different trust and resources. Then 150 services, 75 of them important and 75 unimportant (50/50 ratio), with random resources ($CPU \in [0, 800]$ and $RAM \in [0, 1000]$) are generated so that all nodes are loaded on average to 60%. With the basic self-configuration only one manager is responsible for assigning services. This means that its processing time may take a long time. Such situation can be omitted. With the use of the simultaneous self-configuration, multiple managers are determined to carry out several distribution phases at the same time.

Node Type	CPU (MHz)	Memory (MB)	Trust	Amount (%)
Type 1	200-800	500-1000	0.7-0.9	10
Type 2	500-1500	500-1500	0.3-0.6	50
Type 3	1500-2000	2000-4000	0.4-0.8	30
Type 4	2000-3000	4000-8000	0.4-0.9	10

Table 4.2: For reasons of consistency with the previous results, the same network setting as in Table 4.1 is used.

Hence, the goal is to minimize the self-configuration time. Therefore, it is necessary to use multiple managers during the distribution phase. Moreover, it is measured how many messages are needed to accomplish both algorithms. Our evaluation use the concept of logical clocks [Mat89] to capture clausal ordering of messages (i.e., to provide information as to which message have been sent before a message). Each evaluation scenario has been tested 1000 times with randomly generated networks and the results are averaged. In the following the results of the conducted evaluations are presented.

Self-Configuration Time. The improved simultaneous self-configuration should assure a shorter time to assign services on nodes than the baseline algorithm. This can be shown by varying the number of managers and comparing the behavior of the self-configuration time in both algorithms. Figure 4.12 shows the result of

this experiment, whereas the values on the x-axis stand for time steps and the cumulative amount of assigned services are depicted on the y-axis. A time step is a self-configuration process where all managers distribute simultaneously one of its services (as has been explained in Section 4.3.2). It can be observed that the improved simultaneous self-configuration algorithm with four managers reached the best result followed by the simultaneous version of two managers and the baseline algorithm. To quantify the obtained enhancement of our approach, we calculate the self-configuration time enhancement (STE) as follows (with $ts_b > ts_s$) :

$$STE = \frac{ts_b - ts_s}{ts_b} \tag{4.6}$$

where ts_b means the number of time steps needed to accomplish the baseline algorithm and ts_s is the number of time steps needed to accomplish the simultaneous self-configuration algorithm. As a conclusion to all simulations we have done so

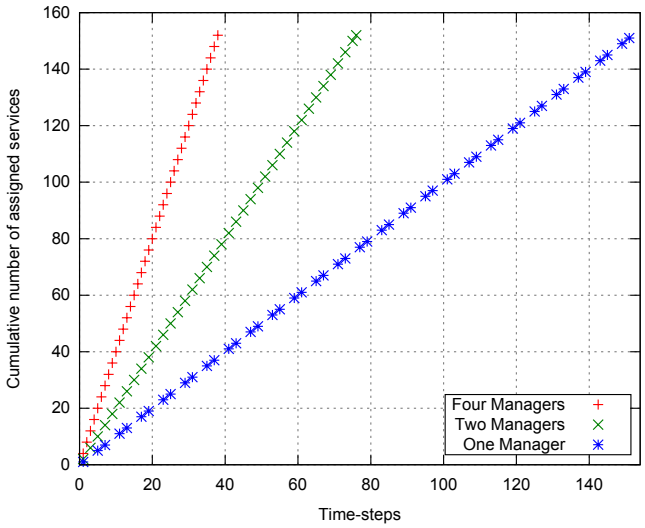


Figure 4.12: Self-configuration using different amount of managers

far (about 1000 runs were evaluated) we can state that STEs of 75% and 50% were achieved respectively to the version of four and the two managers of the simultaneous self-configuration algorithm.

Message Overhead. In the following, the message overhead is evaluated for the proposed self-configuration algorithm. Figure 4.13 shows the result of this experiment. It depicts the number of sent messages on the y-axis. The x-axis stands for the time steps. Figure 4.14 presents a similar information as Figure 4.13, but with a cumulative data to provide a better comparison between the algorithms. The baseline algorithm performs with the least of messages. The number of sent messages

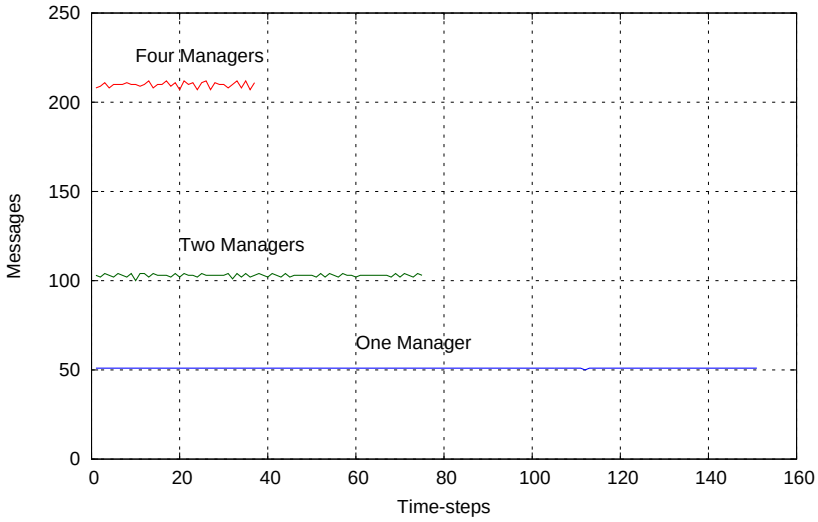


Figure 4.13: The Number of sent messages during the execution

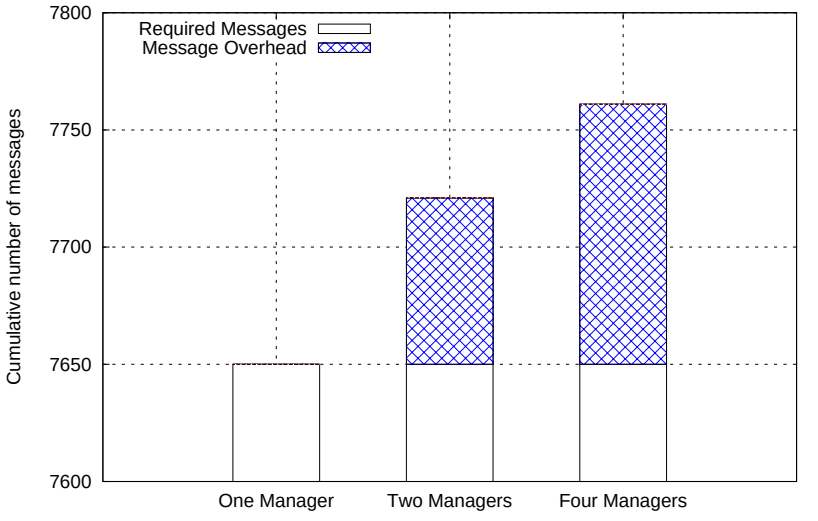


Figure 4.14: The cumulative number of sent messages after the execution

is dependent on the number of managers due to their coordination. Compared to the baseline version, overheads of 71 and 111 messages were produced to the simultaneous algorithm with two and four managers, respectively.

Conclusion of the Results. Concluding, the evaluation result shows that the use of multiple managers in a context of 50 nodes and 150 services improves at least 50% the time to perform the self-configuration while causing a very little deterioration of the message overhead. Compared to the baseline algorithm, overheads of 0.92% and 1,45% were produced to the simultaneous algorithm with two and four managers, respectively. This can be seen as a tolerable trade-off between the message overhead and the self-configuration time and will have more positive impact than negative on the whole system. However, this trade-off depends obviously on the specific size of the network as well as the amount of services running on it, and on the number of managers used in the assignment process. Therefore it is interesting to analyse this trade-off within different context settings. The next section examines this trade-off with respect to different context settings.

Trade-off Exploration. In the following, nine experiments are conducted to further investigate the trade-off between self-configuration time and message overhead. The experiments differ in the in the size of the network η, the amount of services χ and the number of used managers M. The first three experiments examine the trade-off with a fixed $\eta = 100$ but different amounts of χ and M.

- Experiment 1.1: $\eta = 100, \chi = 50, M \in \{1, 2, 4, 8, 16\}$
  (see Figures 4.15 and 4.16)
- Experiment 1.2: $\eta = 100, \chi = 100, M \in \{1, 2, 4, 8, 16\}$
  (see Figures 4.17 and 4.18)
- Experiment 1.3: $\eta = 100, \chi = 150, M \in \{1, 2, 4, 8, 16\}$
  (see Figure 4.19 and 4.20)

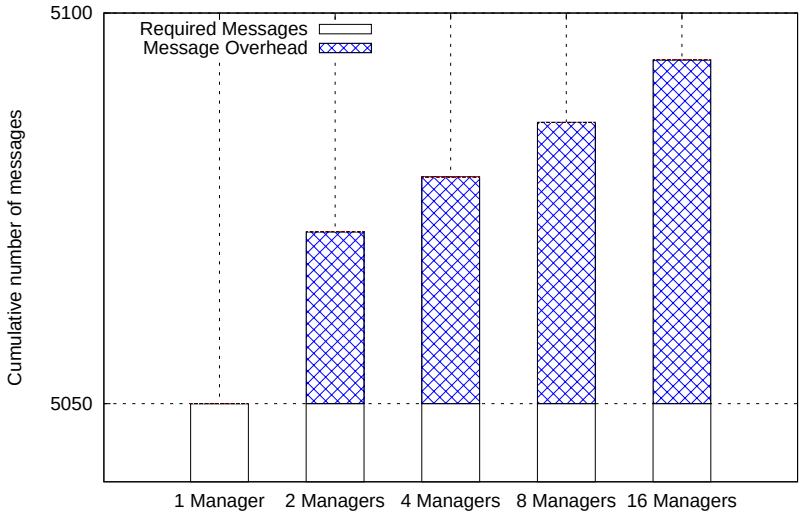


Figure 4.15: Experiment 1.1: $\eta = 100$, $\chi = 50$, $M \in \{1, 2, 4, 8, 16\}$

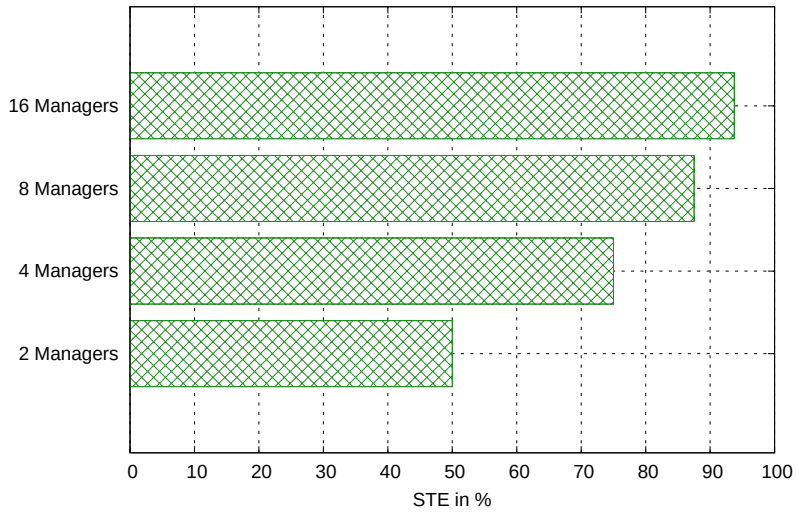


Figure 4.16: Experiment 1.1: $\eta = 100$, $\chi = 50$, $M \in \{1, 2, 4, 8, 16\}$

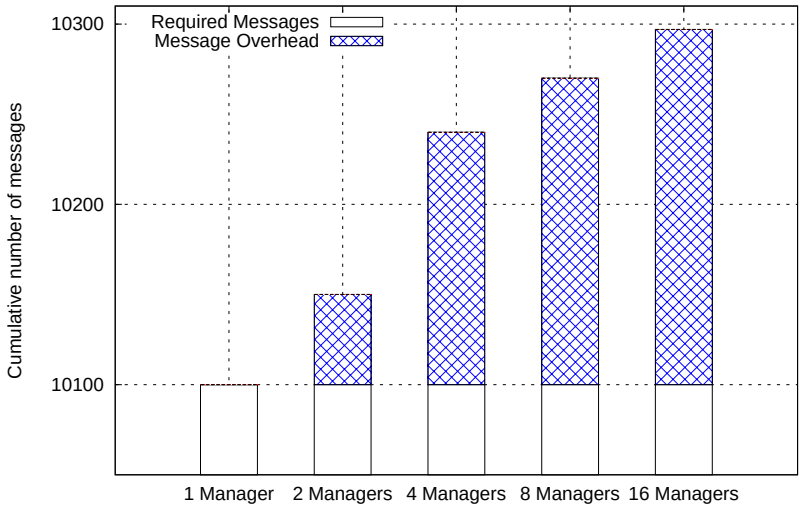


Figure 4.17: Experiment 1.2: $\eta = 100$, $\chi = 100$, $M \in \{1, 2, 4, 8, 16\}$

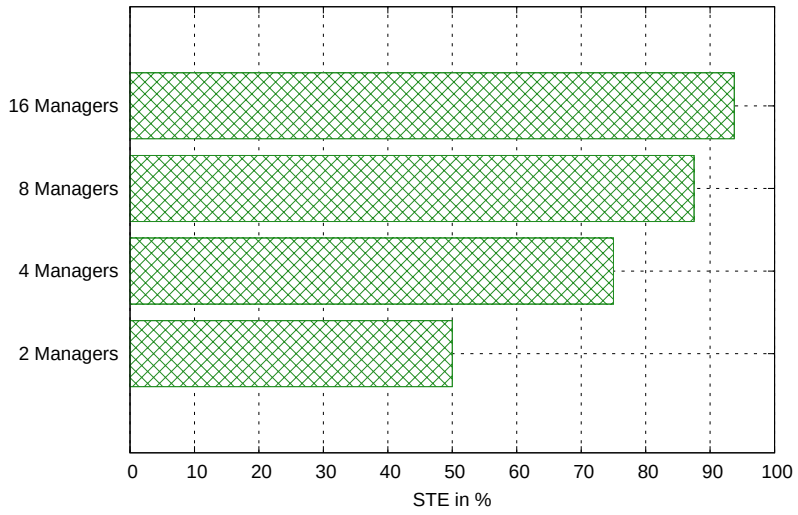


Figure 4.18: Experiment 1.2: $\eta = 100$, $\chi = 100$, $M \in \{1, 2, 4, 8, 16\}$

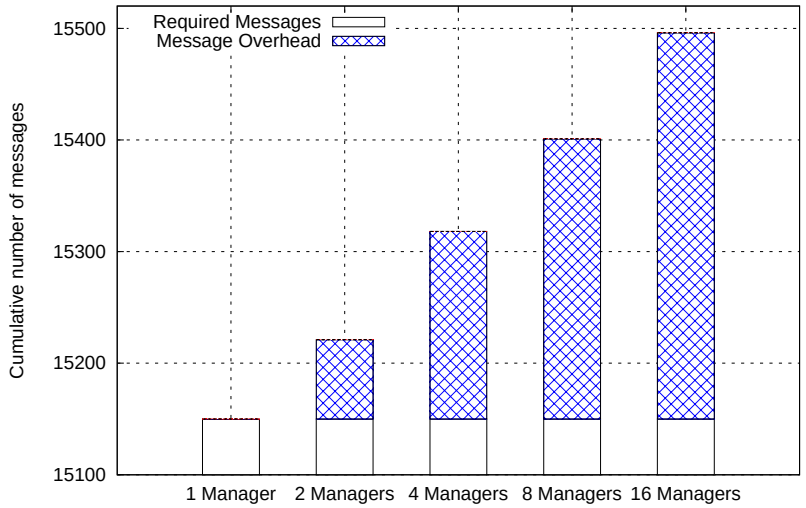


Figure 4.19: Experiment 1.3: $\eta = 100$, $\chi = 150$, $M \in \{1, 2, 4, 8, 16\}$

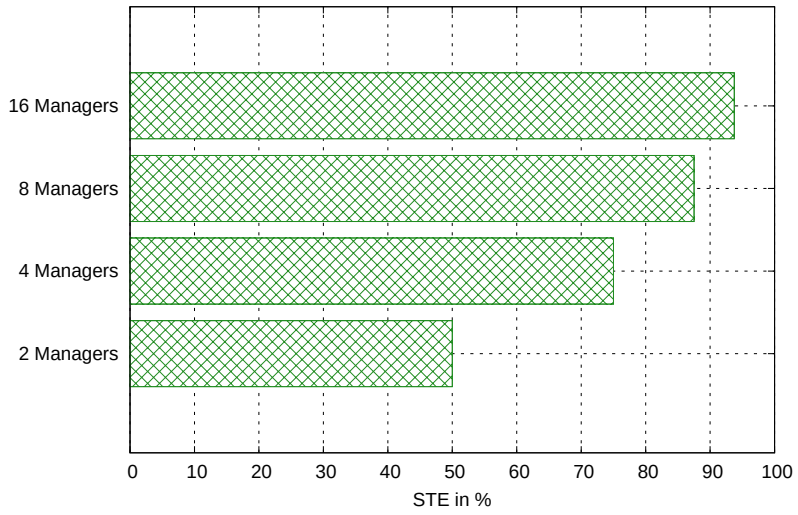


Figure 4.20: Experiment 1.3: $\eta = 100$, $\chi = 150$, $M \in \{1, 2, 4, 8, 16\}$

Experiments 2.1-2.3 consider a fixed network size of $\eta = 150$ but the amount of services χ and the number of managers M have been varied.

- Experiment 2.1: $\eta = 150$, $\chi = 50$, $M \in \{1, 2, 4, 8, 16\}$
  (see Figures 4.21 and 4.22)
- Experiment 2.2: $\eta = 150$, $\chi = 100$, $M \in \{1, 2, 4, 8, 16\}$
  (see Figures 4.23 and 4.24)
- Experiment 2.3: $\eta = 150$, $\chi = 150$, $M \in \{1, 2, 4, 8, 16\}$
  (see Figure 4.25 and 4.26)

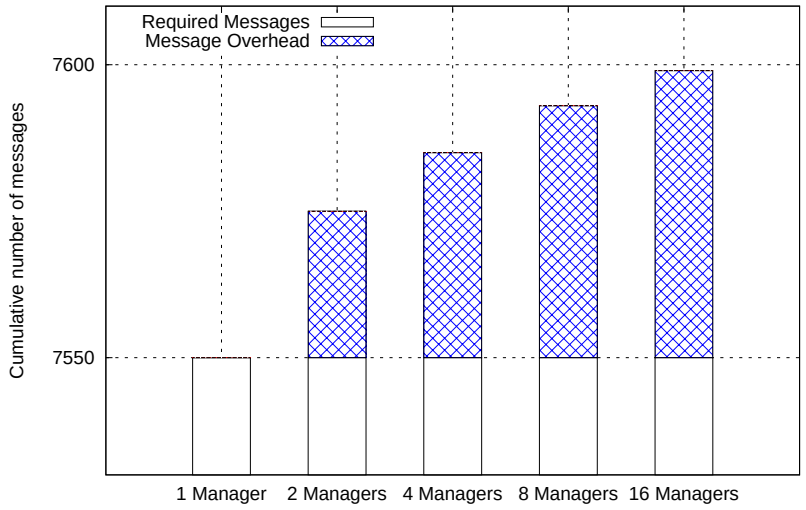


Figure 4.21: Experiment 2.1: $\eta = 150$, $\chi = 50$, $M \in \{1, 2, 4, 8, 16\}$

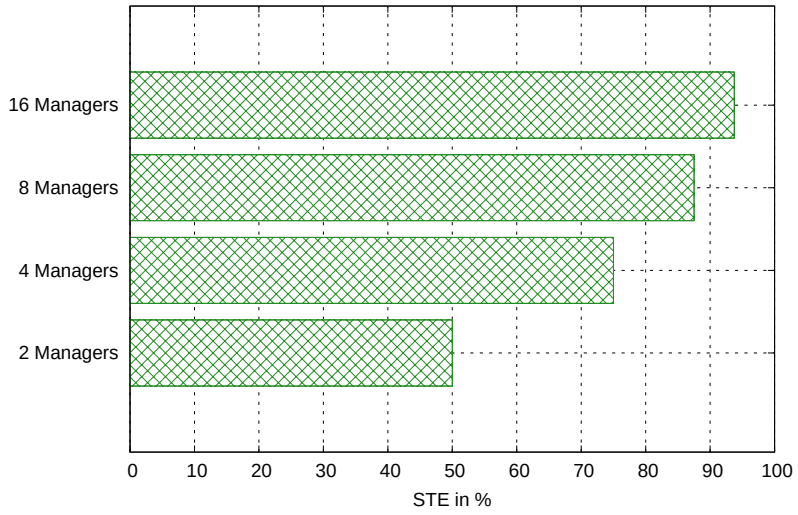


Figure 4.22: Experiment 2.1: $\eta = 150$, $\chi = 50$, $M \in \{1, 2, 4, 8, 16\}$

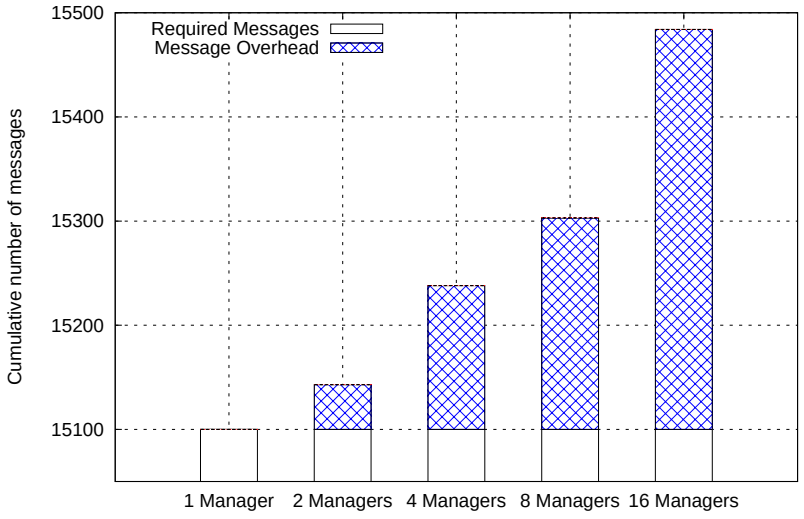


Figure 4.23: Experiment 2.2: $\eta = 150$, $\chi = 100$, $M \in \{1, 2, 4, 8, 16\}$

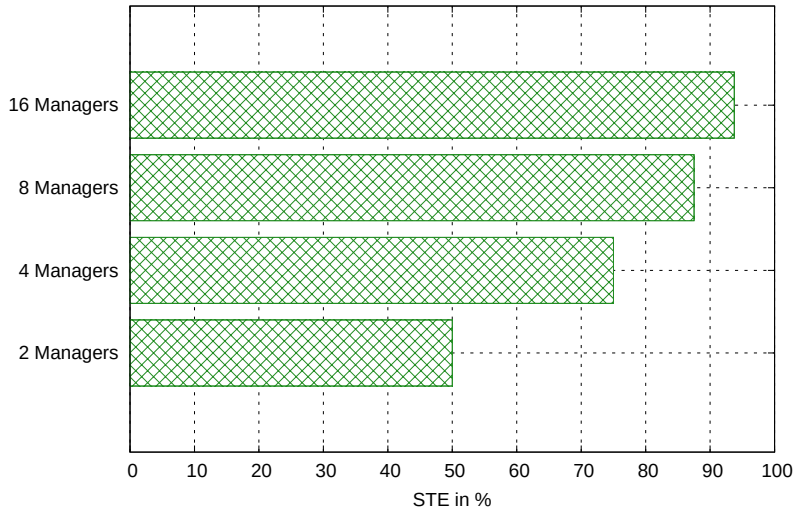


Figure 4.24: Experiment 2.2: $\eta = 150$, $\chi = 100$, $M \in \{1, 2, 4, 8, 16\}$

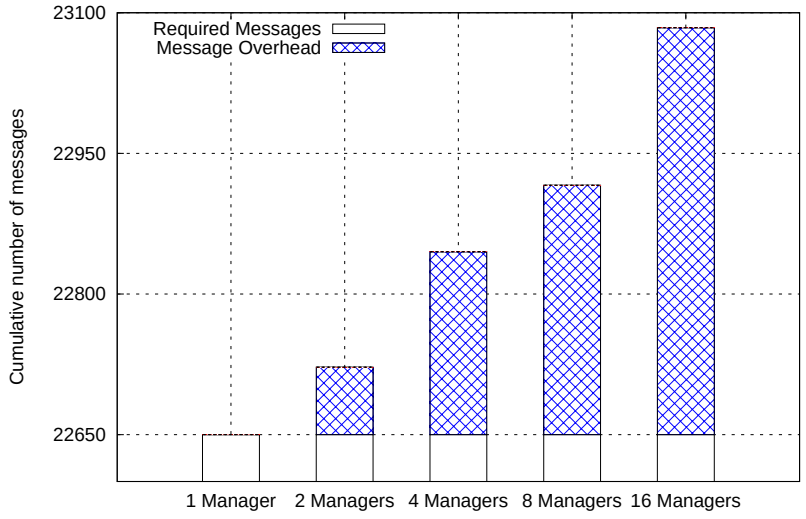


Figure 4.25: Experiment 2.3: $\eta = 150$, $\chi = 150$, $M \in \{1, 2, 4, 8, 16\}$

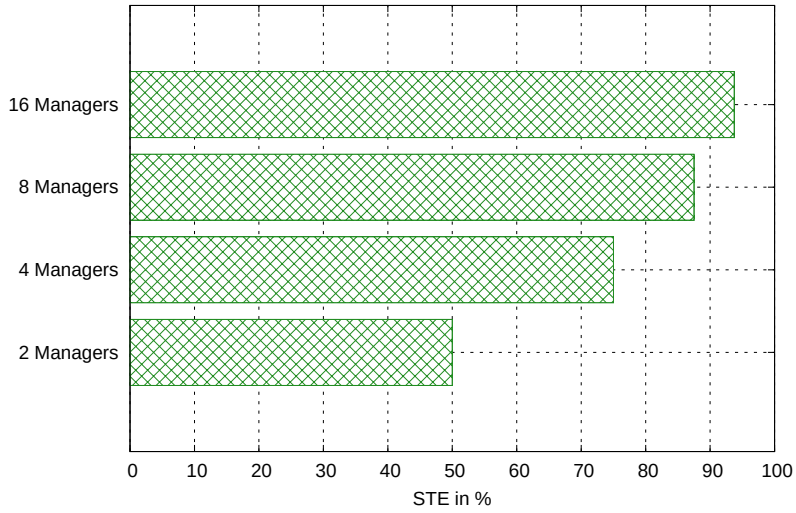


Figure 4.26: Experiment 2.3: $\eta = 150$, $\chi = 150$, $M \in \{1, 2, 4, 8, 16\}$

The following three experiments are similar to the first ones but the the network size is set to $\eta = 200$

- Experiment 3.1: $\eta = 200$, $\chi = 50$, $M \in \{1, 2, 4, 8, 16\}$
  (see Figures 4.27 and 4.28)
- Experiment 3.2: $\eta = 200$, $\chi = 100$, $M \in \{1, 2, 4, 8, 16\}$
  (see Figures 4.29 and 4.30)
- Experiment 3.3: $\eta = 200$, $\chi = 150$, $M \in \{1, 2, 4, 8, 16\}$
  (see Figure 4.31 and 4.32)

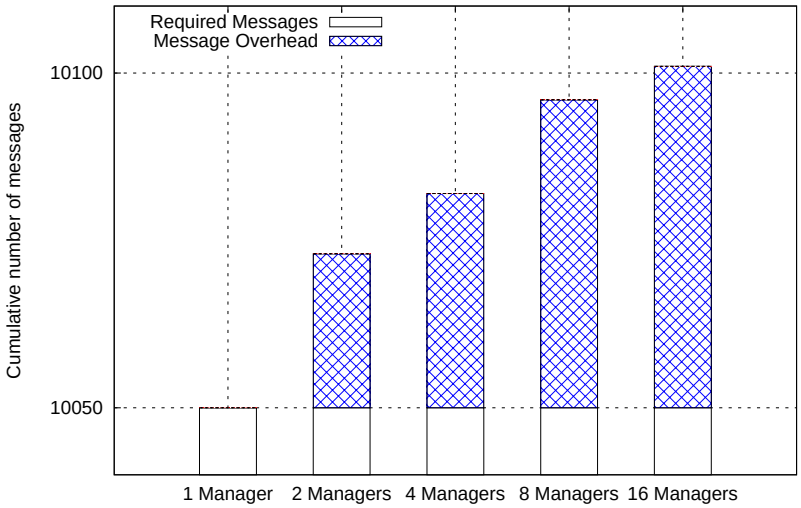


Figure 4.27: Experiment 3.1: $\eta = 200$, $\chi = 50$, $M \in \{1, 2, 4, 8, 16\}$

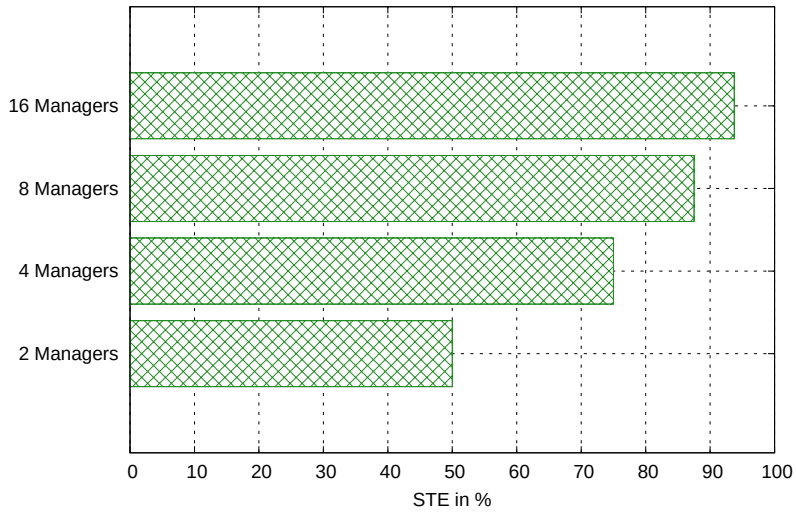


Figure 4.28: Experiment 3.1: $\eta = 200$, $\chi = 50$, $M \in \{1, 2, 4, 8, 16\}$

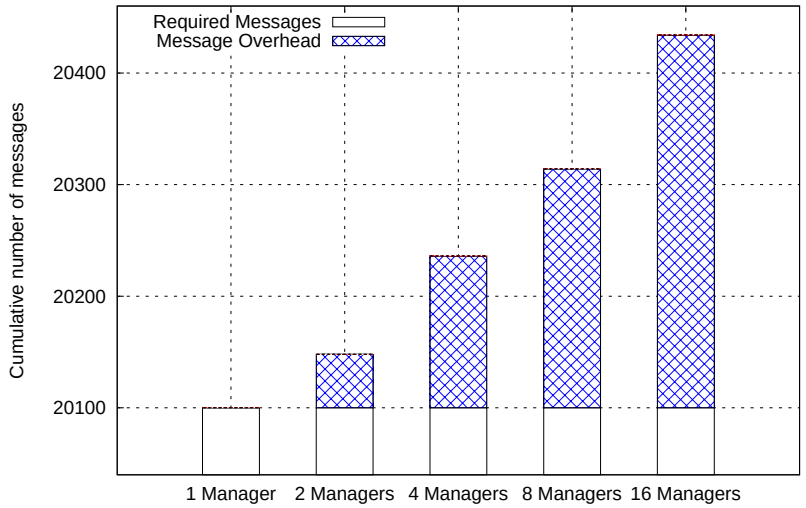


Figure 4.29: Experiment 3.2: $\eta = 200$, $\chi = 100$, $M \in \{1, 2, 4, 8, 16\}$

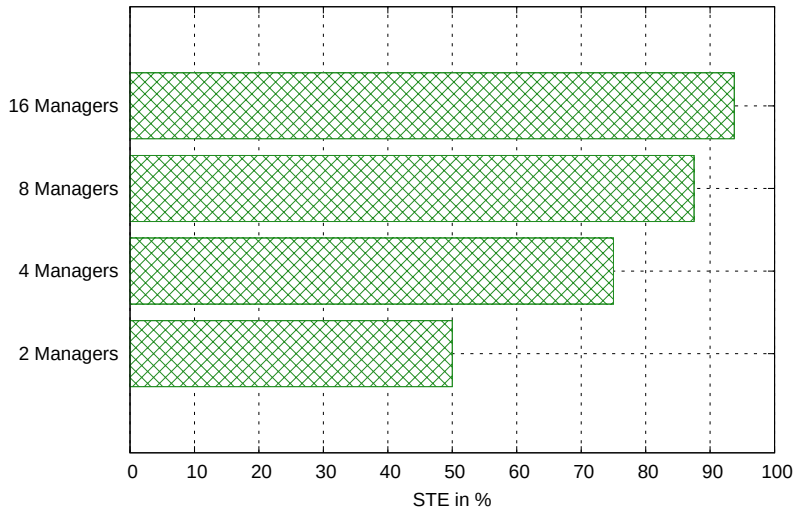


Figure 4.30: Experiment 3.2: $\eta = 200$, $\chi = 100$, $M \in \{1, 2, 4, 8, 16\}$

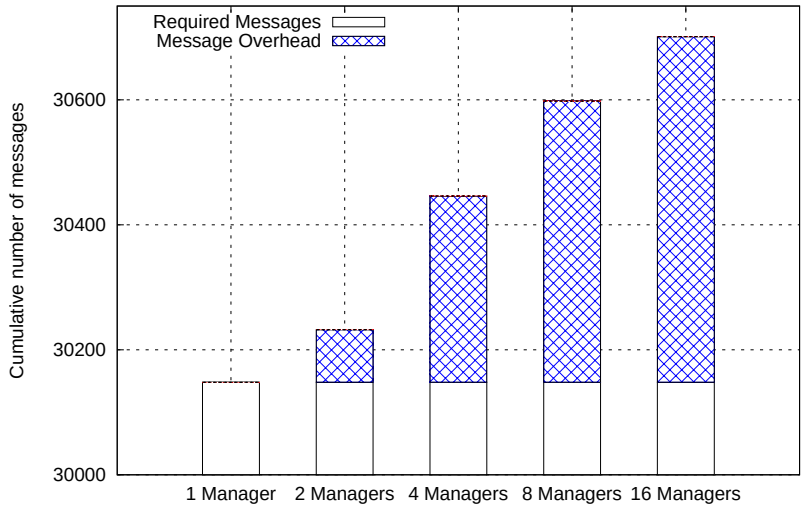


Figure 4.31: Experiment 3.3: $\eta = 200$, $\chi = 150$, $M \in \{1, 2, 4, 8, 16\}$

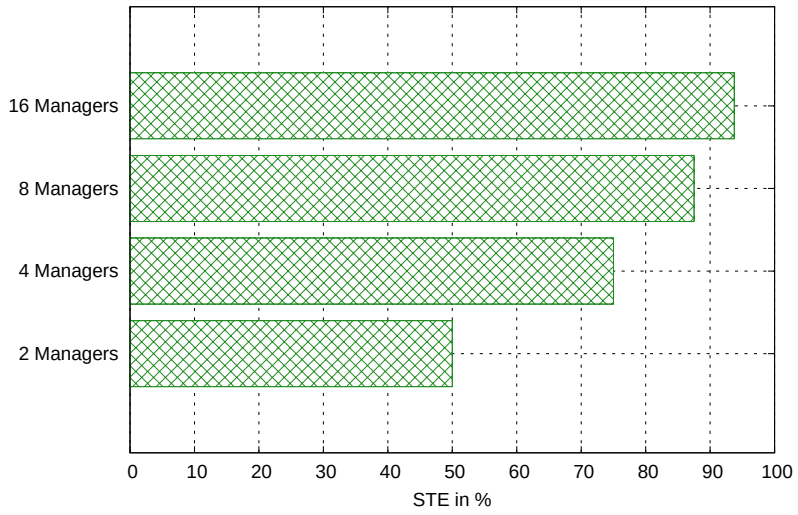


Figure 4.32: Experiment 3.3: $\eta = 200$, $\chi = 150$, $M \in \{1, 2, 4, 8, 16\}$

Conclusion Deduced From Conducting Experiments. The results of the trade-off explorations of Experiments 1.1 - 3.3 are depicted in Figure 4.15 - 4.32. The results attest the introduced simultaneous approach a very good performance in all kind of settings. Indeed, it is characterized by high STEs and low message overhead. The more managers are used in the assignment process, the better will be the improvement of STEs, often with higher although still acceptable overhead below 2.55% in worst context setting (i.e., $\eta = 150$, $\chi = 100$, $M = 16$).

Scalability Behavior. In this part, the scalability behaviour of the simultaneous self-configuration regarding service amount and network size is examined. The number of desired managers is set to 10, while each manager knows all other nodes in the network. As stated in the specification, messages need to be sent to establish the simultaneous self-configuration process. This overhead is evaluated on the basis of the following four experiments:

- **Experiment 1:** $\chi = 100$ services, network size $\eta \in [50, 500]$
- **Experiment 2:** $\chi = 200$ services, network size $\eta \in [50, 500]$
- **Experiment 3:** $\chi = 400$ services, network size $\eta \in [50, 500]$

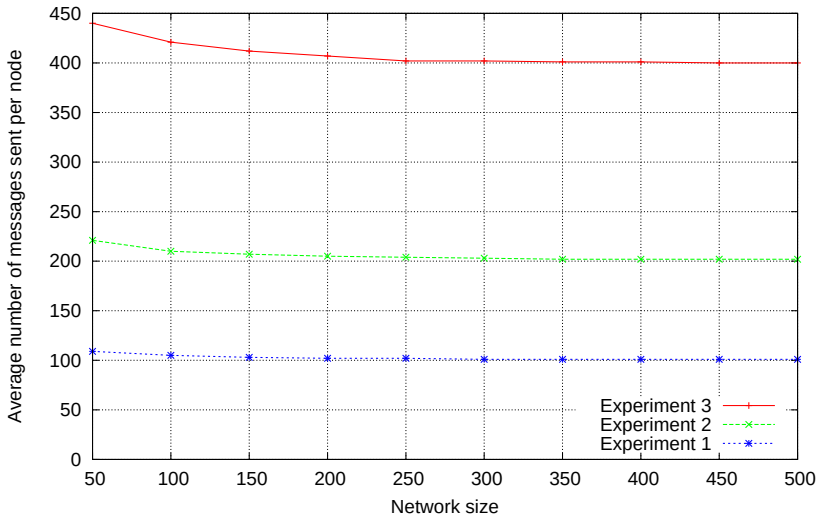


Figure 4.33: Scalability of the self-configuration regarding network size (10 managers)

Please note that all experiments are conducted according to the description given above in Section 4.5.1. Figure 4.33 shows the results of these experiments, whereas the values on the x-axis stand for the network size and the average number of messages by each node is depicted on the y-axis. In all experiments, the results indicate that our self-configuration approach can be classified as being independent from the network size, since nodes do not send more messages within a bigger network. By contrast, the proposed algorithm performs better in a bigger network. The reason for this behaviour is that large networks tend to reduce the competition between managers and thus reduce the message overhead arising from their coordination. In general, the bigger the network, the less coordination between managers is needed. Concluding, thanks to the independence of the message overhead caused by the self-configuration from the network size, our proposed approach seems suitable to be applied in adaptive self-* systems within large number of nodes.

4.6 Conclusions and Future Work

In this chapter, a trust-aware self-configuration algorithm for self-* systems has been proposed. The proposed algorithm extends the well-known Contract Net Protocol [Smi80] by including trust to allow in the system a robust and trustworthy distribution of services on nodes. The trust-based distribution approach was presented in Section 4.3. It aims on the one hand to equally distribute the load of services on nodes as in a typical load balancing scenario and on the other hand to assign services with different importance levels to nodes so that the more important services are assigned to more trustworthy nodes. Performance measurements have been conducted in Section 4.3.4 to evaluate the trust-enhanced self-configuration against the baseline Contract Net Protocol. The results showed that the trust-enhancement mechanisms indeed provide an even better availability for hosting important services than the baseline algorithm. However, self-configuration must take into account the termination time needed for assigning services not only by using one manager but also with multiple managers. Therefore, a way is suggested in Section 4.5 to extend the self-configuration with multiple managers to achieve a simultaneous behaviour of assigning services on nodes. The simultaneous self-configuration have been evaluated within different context settings in Section 4.5.1. The results show an outstanding performance for the simultaneous self-configuration with a decrease in processing time of minimum 50%. For future work, learning mechanisms are planed to investigate the optimal trade-off α between trust and workload. This is very important to further improve the performance of the proposed self-configuration algorithm. One possible solution to

address this issue could be to provide a function that maps the important levels of services with the assortment of α, but as it goes beyond the scope of this work it is not further discussed here.

5

Trust-Aware Self-Optimization

Abstract. In this chapter, we present a self-optimization approach that does not only consider pure load-balancing but also takes into account trust to improve the assignment of important services to trustworthy nodes. Our approach uses different optimization strategies to determine whether a service should be transferred to another node or not. The evaluation results showed that the proposed approach is able to balance the workload between nodes nearly optimal. Moreover, it improves significantly the availability of important services, i.e., the achieved availability was no lower than 85% of the maximum theoretical availability value.

5.1 Introduction

Open distributed systems are rapidly getting more and more complex. Therefore, it is essential that such systems will be able to adapt autonomously to changes in their environment. They should be characterized by so-called self-* properties such as self-configuration, self-optimization and self-healing. The autonomous optimization of nodes at runtime in open distributed environments is a crucial part for developing self-optimizing systems. In this chapter, a trust-aware self-optimization algorithm for self-* systems is presented. It does not only consider

pure load-balancing but also takes into account trust to improve the assignment of important services to trustworthy nodes. The proposed self-optimization approach makes use of different optimization strategies based on trust to determine at runtime whether a service should be transferred to another node or not. The trust definition [SKL+10] adopted for this work is the definition provided by the research unit OC-Trust of the German Research Foundation (DFG) by regarding different facets of trust, as, for example, safety, reliability, credibility and usability. The focus here lies on the reliability aspect. Furthermore, it is assumed that a node can not realistically assess its own trust value because it trusts itself fully. Therefore, the calculation of the trust value in this work must be done with the previously introduced trust metrics presented in chapter 3.4. With trust information, nodes of a system have a reference about which nodes to cooperate with, and this is important for self-optimizing systems. The chapter offers as contribution the following aspects:

(i) a decentralized self-optimization algorithm for load balancing taking into account trust — respectively reliability — to increase the robustness of important services in open distributed environments (see Sections 5.3 and 5.4),

(ii) a formal description of the optimization strategies to determine at runtime whether a service should be transferred to another node or not (see Section 5.5), and

(iii) a set of extensions for the basic algorithm to further improve its performance time in case of multiple simultaneous requests (see Section 5.6)

All aspects are evaluated and discussed with respect to a toolkit based on the TEM [ASM+13], a trust-enabling middleware for building real-world distributed Organic Computing systems. Section 5.7 provides evaluation results of the proposed self-optimization algorithm and demonstrate the benefits of the proposed extensions. Finally, the chapter is closed with a conclusion and future work in Section 5.8.

5.2 Related Work

A lot of papers have been published to deal with the assignment problem of services on nodes, either to achieve a static or dynamic load balancing [KAA+13, BKL00, PM15, SKC15, AUNDFMGS+15, ABAS16]. In most existing algorithms, the consideration of the trustworthiness of nodes has been neglected so far. For instance, the work of Rao et al. [RLS+12] proposes several methods for solving the

load balancing problem in distributed systems. One of these methods, called one-to-one, is similar to our approach: two nodes are picked at random. Then, a virtual server transfer is initiated if one of the nodes is heavy and the other is light. Their method, however, does not consider how the availability of important services may be improved, and does not distinguish between trustworthy and untrustworthy nodes. Bittencourt et al. [BMCB05] presented an approach to schedule processes composed of dependent services onto a grid. This approach is implemented in the Xavantes grid middleware and arranges the services in groups. It has the drawback of a central service distribution instance and therefore a single point of failure can occur. In [LSJB11], two different self-optimization algorithms for LTE networks are presented. One of these algorithms, called Load Balancing in Downlink LTE networks, is similar to our approach. The authors try to shift the virtual load of overloaded cells to less loaded adjacent cells by changing the virtual cell borders. The virtual load is modeled as the sum of resources needed to achieve a certain QoS for all active user equipments. Matrix [WZL+14] is another approach to combine load optimization with data-aware scheduling. The authors propose to apply adaptive work stealing techniques to achieve load balancing in distributed many-tasks computing environment. Tasks are organized in queues based on their size and locations. Then, a ZHT is used to submit tasks to idle schedulers and to monitor the execution progress of tasks in a scalable way. Whenever a scheduler has no more tasks, it communicates with other heavy-loaded schedulers to receive new tasks. Their approach does not take the priority of different service classes into account. In [SMB+09], the authors presented a receiver-initiated optimization algorithm that automatically balances the workload of nodes in distributed computing environments. It is implemented in the OCμ middleware. In their algorithm, services can be relocated or transferred to other nodes to balance the resource consumption among nodes. Moreover, it takes the trust constraints of nodes into account to transfer important services only to trustworthy nodes. However, it is based on the unrealistic assumption that all nodes have the same resource capacity. Contrary to this work, our approach is able to work with heterogeneous capacities. More precisely, we are interested in a dynamic receiver-initiated [DLEJ86] self-optimization algorithm (i.e., since services are assumed not to be stolen from other nodes) that has neither a central control nor complete knowledge about the system. The algorithm must not only consider pure load-balancing but also takes into account trust to improve the assignment of important services to trustworthy nodes. And all this at runtime. Parts of the content of this chapter have been published by the author in the following conference and journal:

- **[MKU14b]:** Nizar Msadek, Rolf Kiefhaber, and Theo Ungerer. *A Trust- and*

Load-Based Self-Optimization Algorithm for Organic Computing Systems. SASO 2014: Proceedings of the 8th International Conference on Self-Adaptive and Self-Organizing Systems, pages 177-178, London, England, IEEE Computer Society, 2014. 2014.

- **[MKU15b]:** Nizar Msadek, Rolf Kiefhaber, and Theo Ungerer. *Trustworthy Self-Optimization in Organic Computing Environments*. In ARCS 2015: Proceedings of the 28th International Conference on Architecture of Computing Systems, pages 123-134, Porto, Portugal, Springer 2015.
- **[MU17b]:** Nizar Msadek and Theo Ungerer. *Trustworthy Self-Optimization for Organic Computing Environments Using Multiple Simultaneous Requests*. In JSA 2017: Journal of Systems Architecture, issn 1383-7621, `http://www.sciencedirect.com/science/article/pii/S1383762117301388`

5.3 Basic Idea of the Self-Optimization Algorithm

A distributed system consisting of a set of n nodes $\mathcal{N} = \{n_1, n_2.., n_n\}$ is considered, where each node can interact with each other through a set of application messages. They can optimize at runtime the assignment of services in the network by transferring their own services to other nodes. Suppose that node j at a certain point during runtime sends an application message to another node i. It appends onto the outgoing message (a) its trust in node i (b) its current workload and (c) some information (i.e., importance level and consumption) about services, which are running on it. Based on this information node i decides which of the following optimization strategies should be performed:

5.3.1 No Optimization

- **Description:** The workload between nodes is well balanced and their trust values are similar enough.
- **Discussion:** This is the simplest case that can happen between nodes. Both of them are well optimized in terms of trust and workload.
- **Solution:** Nothing will happen

5.3.2 Load Optimization

- **Description:** Trust of nodes is similar enough but their workload is unbalanced.

- **Discussion:** This strategy aims to find a pure load balancing between nodes since their trust is similar enough.
- **Solution:** Services are transferred in order to balance the workload between the nodes. Then, two cases are distinguished: (a) either the workload of i is higher or (b) the workload of j is higher. In the case of (a), node i balances the workload of the nodes by transferring a subset of its services to j. Otherwise, node i sends an alert message to j together with all information which are necessary for the optimization. Case (a) will be then triggered on side of j.

5.3.3 Trust Optimization

- **Description:** The workload between nodes is well balanced but their trust values differ significantly. In this case important services might run on untrustworthy nodes and are prone to fail.
- **Discussion:** This strategy aims to use particularly trustworthy nodes for important services. Therefore, important services have to be relocated to more trustworthy nodes and unimportant services to less trustworthy nodes. Furthermore, the overall workload resources between nodes should still be well-balanced.
- **Solution:** By this strategy, we distinguish between two cases: (a) either i is more trustworthy than j or (b) j is more trustworthy than i. If (a), then i swaps its unimportant services for important services of j. In the case of (b), node i swaps its important for unimportant services of j. Note that the load consumption between important and unimportant services should be similar to keep the load-balancing property in both nodes satisfied.

5.3.4 Trust and Load Optimization

- **Description:** Trust of nodes differs significantly and their workload is unbalanced.
- **Discussion:** This strategy aims at workload balancing with additional consideration of the services' priority, i.e. to avoid hosting important services on untrustworthy nodes.
- **Solution:** Four cases are distinguished: (a) either the workload of i is higher and i is more trustworthy than j, (b) the workload of i is higher but j is more trustworthy, (c) the workload of j is higher but it is less trustworthy than i, or finally (d), the workload of j is higher and it is also more trustworthy than j.

In the case of (a), node i balances the workload of load by transferring only unimportant services to j. If there are no unimportant services available, then no optimization is done. The rationale for this step is that there is a trade-off between trust and workload. Improving one of these criteria will typically deteriorate the other. In the case of (b), node i balances the workload by transferring only important services to j. Just as the case of (b), no optimization is done, if there are no available unimportant services. In other cases (i.e., c and d), node i sends an alert optimization message to j to piggy-back information necessary for self-optimization. Depending on the situation, case (a) or (b) will be then triggered on side of j.

5.4 Metrics and Notions

Since it is very complex to address the self-optimization problem in its full generality, we make some simplifying assumptions. Firstly, we assume that the load of a service is stable (or can otherwise be predicted) over the time interval it takes for the self-optimization algorithm to operate. Secondly, we assume there is only one bottleneck resource we are trying to optimize for. Let w_i denote the workload of a node i, where w_i represents the sum of the resource consumptions of all services running on node i (see Formula 5.1).

$$w_i = \sum_{s \in \mathcal{S}_i} c_s, \text{ with } 0 \leq w_i \leq C_i^{max}. \tag{5.1}$$

It is to note that c_s means the resource consumption of a service s. The maximum resource capacity of a node i is denoted by C_i^{max} and its set of services by S_i. Moreover, we divide services $\mathcal{S}_i$ into two sets based on their importance levels:

- $\mathcal{S}_i^{imp}$: Set of important services (running on node i), which are necessary for the functionality of the entire system.
- $\mathcal{S}_i^{unimp}$: Set of unimportant services (running on node i), which have only a low negative effect on the entire system if they fail.

Then, considering only the context of pure load optimization, our goal is to balance the workload between nodes. Let us assume two nodes, i and j: node i is underloaded. However, node j is overloaded and its task is to balance the workload by service transfers to i. Thus, as you can see in Figure 5.1, j transfers its services whose cumulative resource consumption is close enough to $\frac{|w_j - w_i|}{2}$ (optimal balancing). Although this simple idea seems to make a lot of sense, its drawback arises when the resource capacities of nodes are significantly different (see Figure 5.2).

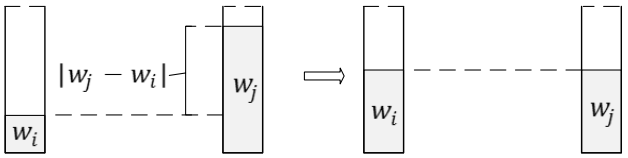


Figure 5.1: Simple load optimization method

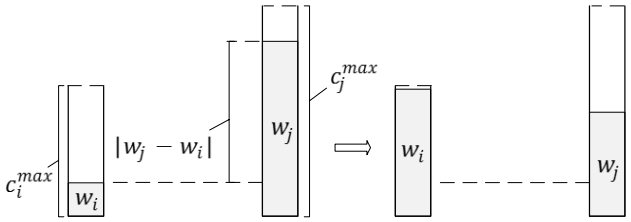


Figure 5.2: Nodes still unbalanced due to their different resource capacities

Therefore, we introduce a new optimal theoretical workload O_i, which should serve as a target reference point for every node. The node which surpasses this reference point ($w_i > O_i + \delta_{tol}$) is considered to be overloaded, otherwise it is underloaded ($w_i < O_i - \delta_{tol}$) or balanced ($|O_i - w_i| \leq \delta_{tol}$), where a δ_{tol} is a tolerable threshold and represents the quality to reach the perfect workload. The optimal theoretical workload of a node i is calculated using Formula 5.2. Since w_i is normalized in a different capacity than w_j, we must first divide the sum of workload $w_i + w_j$ by the sum of capacity $c_i^{max} + c_j^{max}$ to obtain the optimal theoretical workload per one unit capacity, which will be then multiplied by c_i^{max}. Furthermore, each node has an individual trust value calculated based on the previously introduced trust metrics presented in chapter 3.4. Recall, the trust value $t_i(j)$ represents the subjective trust of node i in node j and will always range between 0 and 1. The value of 0 means that i does not trust j at all while a value of 1 stands for complete trust. Two nodes i and j are considered to have a similar trust behavior if $|t_i(j) - t_j(i)| \leq \gamma_{tol}$, where γ_{tol} is a tolerable threshold and reflects the quality to achieve a good trust similarity between nodes.

$$O_i = \frac{w_i + w_j}{c_i^{max} + c_j^{max}} c_i^{max} \tag{5.2}$$

5.5 The Algorithm in Detail

The algorithm proposed in this section represents a best-effort approach to improve the assignment of services on nodes so as to satisfy both workload and trust constraints. It is used to solve this problem in a distributed manner. We assume that nodes of the network do not know the workload of others until they receive a message from a node with information about that. The workload of nodes also might change over time. We further assume that a node can not assess its own trust value, but is rated by other nodes. Therefore, its trust value must be calculated from the neighbor nodes of the network (see [KJMU13] for more details). Note that the trust of nodes might also change over time. Again we are considering two nodes i and j, where j sends an application message m_j to i, on which it piggybacks the following additional information:

- $\mathcal{S}_j^{unimp}$: Set of less important services running on node j
- $\mathcal{S}_j^{imp}$ **:** Set of important services running on j
- $t_j(i)$: Current trust value of j in i
- w_j: Current workload value of j
- c_j^{max}: Maximum resource capacity of j

Based on this information node i decides which optimization strategy should be performed. In the following we consider all possible decisions a node i has to make:

5.5.1 No Optimization

- **Formal description:** $|t_i(j) - t_j(i)| \leq \gamma_{tol}$ and $|O_i - w_i| \leq \delta_{tol}$
- **Solution:** Nothing will happen

5.5.2 Load Optimization

- **Formal description:** $|t_i(j) - t_j(i)| \leq \gamma_{tol}$ and $|O_i - w_i| > \delta_{tol}$
  - **Case a:** $w_i > O_i$ **and** $w_j < O_j$
    Node i balances the workload by transferring some of its services to j, regardless of whether they are important or not since the trust of nodes is similar. Firstly, it determines $\Psi_{i,j}$ (see Formula 5.3 and 5.4) as a set of services that could be selected to balance the workload of nodes. Note that $\mathcal{C}(I_s)$ represents the consumption function of a set of services I_s and is calculated by the sum of all its service consumptions.

$$\Psi_{i,j} = \{I_s \mid I_s \subseteq (\mathcal{S}_i^{imp} \cup \mathcal{S}_i^{unimp}) : max\, \mathcal{C}(I_s) \text{ and } \mathcal{C}(I_s) \leq (O_j - w_j) \text{ and } 0 < \mathcal{C}(I_s) \leq (w_i - O_i)\} \tag{5.3}$$

$$\mathcal{C}(I_s) = \sum_{s \in I_s} c_s \tag{5.4}$$

If $\Psi_{i,j}$ is empty, then no optimization is done. Otherwise i transfers $\Psi_{i,j}$ to j.

– **Case b:** $w_i < O_i$ **and** $w_j > O_j$
Since services are assumed not to be stolen from other nodes, node i sends an alert message to j to piggy-back information necessary for self-optimization as described above. Then, case (5.5.2-a) will be triggered but on the side of j.

5.5.3 Trust Optimization

- **Formal description:** $|t_i(j) - t_j(i)| > \gamma_{tol}$ and $|O_i - w_i| \leq \delta_{tol}$
  – **Case a:** $t_j(i) > t_i(j)$
  In this case i determines $\Psi_{i,j}$ (see Formula 5.5) as a set of unimportant services (i.e., with the maximum load consumption) that could be exchanged for important services of j so that the difference of their load consumption never exceeds $\mathcal{C}_{tol}$ to keep the load-balancing property in both nodes satisfied.

$$\Psi_{i,j} = \{I_s \mid I_s \subseteq \mathcal{S}_i^{unimp}, \exists J_s \subseteq \mathcal{S}_j^{imp} : max\, \mathcal{C}(I_s) \text{ and } |\mathcal{C}(I_s) - \mathcal{C}(J_s)| \leq \mathcal{C}_{tol} \text{ and } (\mathcal{C}(I_s) + w_j) \leq c_j^{max}\} \tag{5.5}$$

Then, after transferring $\Psi_{i,j}$, node i sends an alert optimization message to j (i.e., including all information which are necessary for the optimization) in order to trigger case (5.5.4-b) on side of j. Note that the execution of this step aims to balance again the workload between the nodes.

  – **Case b:** $t_j(i) < t_i(j)$
  In contrast to case (5.5.3-a), $\Psi_{i,j}$ is determined only from important services (see Formula 5.6), since j is more trustworthy than i. Then, i sends an alert optimization message to j in order to trigger case (5.5.4-a) on side of j.

$$\Psi_{i,j} = \{I_s \mid I_s \subseteq \mathcal{S}_i^{imp}, \exists J_s \subseteq \mathcal{S}_j^{unimp} : max\,\mathcal{C}(I_s) \text{ and } |\mathcal{C}(I_s) - \mathcal{C}(J_s)| \leq \mathcal{C}_{tol} \text{ and } (\mathcal{C}(I_s) + w_j) \leq c_j^{max}\} \tag{5.6}$$

5.5.4 Trust and Load Optimization

- **Formal description:** $|t_i(j) - t_j(i)| > \gamma_{tol}$ and $|O_i - w_i| > \delta_{tol}$
  - **Case a:** $w_i > O_i$ **and** $w_j < O_j$ **and** $t_j(i) > t_i(j)$
    Node i balances the workload only by transferring unimportant services to j (i.e., due to the fact that i is more trustworthy than j). It determines $\Psi_{i,j}$ as a set of only unimportant services that could be selected to balance the workload of nodes (see Formula 5.7). Then, i transfers $\Psi_{i,j}$ to j.

$$\Psi_{i,j} = \{I_s \mid I_s \subseteq \mathcal{S}_i^{unimp} : max\,\mathcal{C}(I_s) \text{ and } \mathcal{C}(I_s) \leq (O_j - w_j) \text{ and } 0 < \mathcal{C}(I_s) \leq (w_i - O_i)\} \tag{5.7}$$

  - **Case b:** $w_i > O_i$ **and** $w_j < O_j$ **and** $t_j(i) < t_i(j)$
    Since j is more trustworthy than i, $\Psi_{i,j}$ will be determined only from important services (see Formula 5.8). Then, just as the case of (5.5.4-a), if $\Psi_{i,j}$ is empty, no optimization is done. Otherwise i transfers $\Psi_{i,j}$ to j.

$$\Psi_{i,j} = \{I_s \mid I_s \subseteq \mathcal{S}_i^{imp} : max\,\mathcal{C}(I_s) \text{ and } \mathcal{C}(I_s) \leq (O_j - w_j) \text{ and } 0 < \mathcal{C}(I_s) \leq (w_i - O_i)\} \tag{5.8}$$

  - **In other cases:**
    Node i sends an alert message to j (i.e., including all information which are necessary for the optimization). Depending on the situation, case (5.5.4-a or 5.5.4-b) will then be triggered on the side of j.

5.6 Multiple Simultaneous Requests

In the evaluation, we have shown that the basic self-optimization algorithm presented in Section 5.5 led to good performance in terms of trust and workload, but we think that there is a room for improvement with the mechanism presented in this section. Therefore, we analyze now a network situation consisting of multiple simultaneous requests which are addressed to a single node to trigger the self-optimization process. Figure 5.3 gives an overview of this situation. Let n_i denote

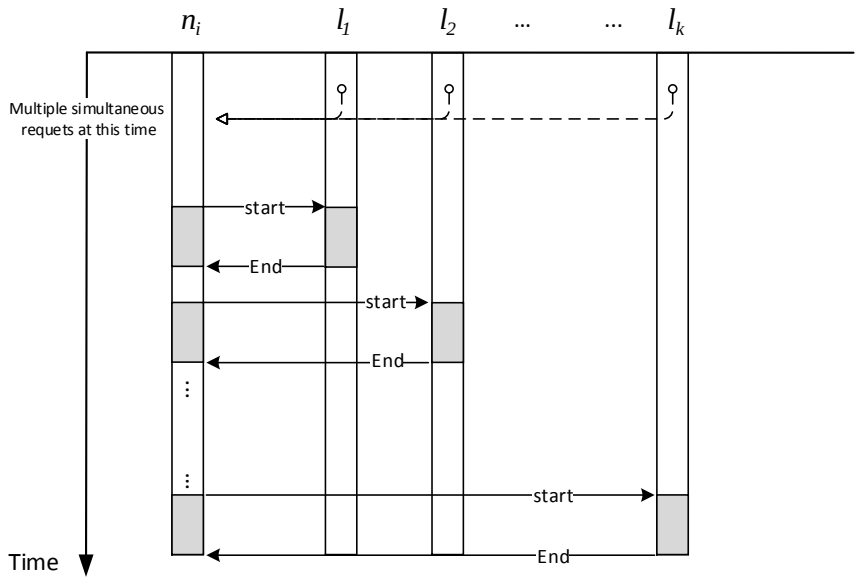


Figure 5.3: Current execution of the basic algorithm

the node that receives the requests and let be $\mathcal{L}^i = \{l_1, l_2, ..., l_k\}$ the set of requesters considered by n_i. We first start with the description of the environment of n_i that has full information about its requesters. It can easily determine the set of potential service transfers Ψ_{n_i,l_j} for each requester $l_j \in \mathcal{L}^i$, using the equations cited in Section 5.5, depending on the current situation of nodes. In the basic approach, as shown in Figure 5.3, n_i optimizes itself with the requesters one after another in a random way without having preference for those that have many potential service transfers. By this means, the overall optimization in the system might take a long time before a large amount of services are transferred, particularity with a growing number of requesters. As a result, too much time can be spent in the whole system to get better optimized nodes. Our goal is to reduce this time by transferring the maximum amount of services as early as possible at runtime. Two approaches can be used to handle this problem.

5.6.1 Selective Request Handling

The first approach is called *selective request handling* because it always allows n_i to select the best requester to perform the optimization. We make use of two parameters in our approach, namely X and S_Ψ. The first parameter X is initialized as the

Algorithm 1 Node n_i:

1: $X \leftarrow \mathcal{L}^i$ ▷ initialize X as the set of all involved requesters
2: S_Ψ = nil ▷ S_Ψ is initialized as empty list of fixed size $|\mathcal{L}^i|$
3: **for** $x \in X$ **do**
4: calculate $|\Psi_{n_i,x}|$ and append it to S_Ψ
5: **end for**

6: **while** $X \neq \emptyset$ **do**
7: select from S_Ψ the requester x with:
8: $\{x | \exists x \in X : |\Psi_{n_i,x}|$ is max and $|\Psi_{n_i,x}| > 0\}$
9: **if** no requester with such a property exists **then**
10: **exit**
11: **else**
12: x perform an optimization with n_i
13: remove x from X
14: **end if**
15: **end while**

set of all involved requesters — in our case always $\mathcal{L}^i$ — and S_Ψ is an empty list of fixed size $|\mathcal{L}^i|$ used to store the potential number of service transfers. The basic idea behind the algorithm is: Whenever n_i receives multiples requests, it calculates the number of service transfers for every requester and applies an optimization with the requester whose services are most among the remaining requesters in X. If there is no requester with such a property, nothing will be done, as the nodes are already optimized. Otherwise, the found requester is removed and this process is repeated until all requesters are processed. In Algorithm 1, the above described algorithm is formalized as pseudo-code. This approach is very simple – and even in the worst case it is at least never worse than doing optimization with random selection – but the optimization output might be suboptimal regarding the overall self-optimization time due to its sequential processing. Therefore, we are interested in the second approach to provide a solution which supports parallelism through the optimization of requesters.

5.6.2 Parallel Request Handling

While in the first approach we match n_i to a single requester to perform the optimization process, in this approach we consider a parallel optimization between requesters that work together to maximize the number of service transfers, as shown in Figure 5.4. This has the benefit to further decrease the optimization time in the whole system. However, nodes in our system have different trust and workload

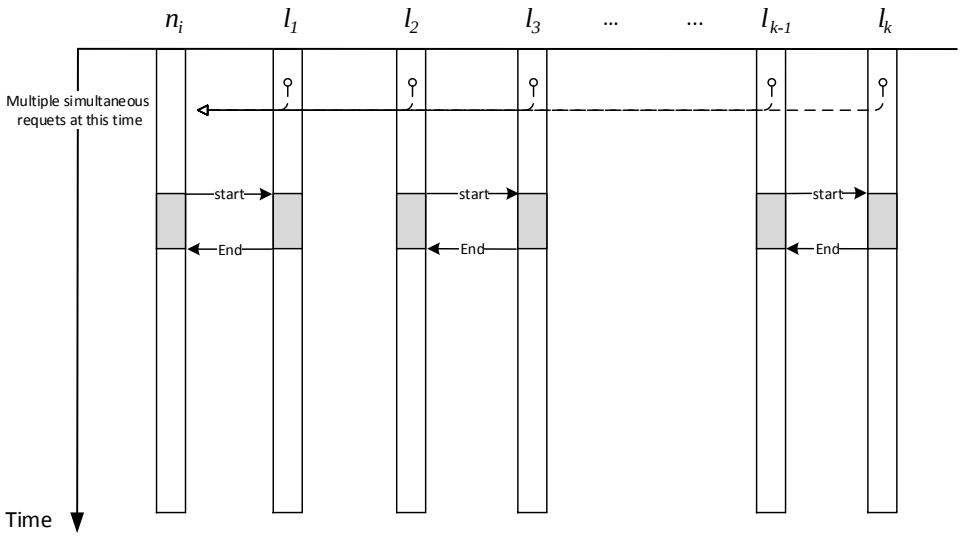


Figure 5.4: Simplified representation of the parallel request handling

values and some of them can transfer more services with one than others. Therefore, an important aspect for n_i is the formation of pairs between nodes — to apply the optimization algorithm in pairs and parallel — but in a way that the number of service transfers will be maximized in the system in order to deliver better results. Algorithm 2 shows the proposed mechanism formalized as pseudo-code.

At the beginning, we initialize two parameters X and T_Ψ. The first parameter $X = \{n_i\} \cup \mathcal{L}^i$ represents the set of all nodes involved in the multiple requests, whereas the second parameter T_Ψ stands for an integer matrix of size $|X| \times |X|$, which we use to store the number of service transfers between nodes. Again, we say that x can optimize itself better with y than z, if and only if $|\Psi_{x,z}| \leq |\Psi_{x,y}|$ with $y \neq z$. Then, the algorithm is split into two phases, the first of which is similar to the selective request handling, but we now allow to calculate the number of service transfers between any two nodes in X. Intuitively, reflexive suitability values such as $\Psi_{x,x}$ are not computable in this phase, simply because it is not allowed that a node is optimizing itself. Afterwards, the algorithm enters in its second phase exploring pairs having at least a service transfer of one and maximizing at the same

Algorithm 2 Node n_i:

1: $X \leftarrow \{n_i\} \cup \mathcal{L}^i$ ▷ initialize X as the set of all involved nodes

2: $T_\Psi \leftarrow$

	n_i	l_1	...	l_k
n_i	0	0	...	0
l_1		0	$\ddots$	0
$\vdots$			0	0
l_k				0

▷ is an empty lookup table of size $|X| \times |X|$

Phase 1

3: **for** $x \in X$ **do**
4: **for** $y \in X \setminus \{x\}$ **do**
5: calculate $|\Psi x, y|$ and append it to T_Ψ
6: **end for**
7: **end for**

Phase 2

8: **while** two nodes remain in X **do**
9: select from T_Ψ the pair (x, y) with:
10: $\{(x, y) | \exists x, y \in X : |\Psi_{x,y}|$ is max and $|\Psi x, y| > 0\}$
11: **if** no pair with such a property exists **then**
12: **exit**
13: **else**
14: x and y become engaged to perform the optimization
15: remove x and y from X
16: **end if**
17: **end while**

time the number of service transfers. If there is no pair with such a property, the algorithm terminates. Otherwise, the found pair becomes engaged to perform the optimization process. Then, the pair is finally removed from the set of X. The while loop continues until there are no more pairs to perform the optimization process. To demonstrate the proposed algorithm an example is discussed.

- **Example:** In this example, an instance of parallel request handling involving five requesters is considered, with $X = \{n_i, l_1, l_2, l_3, l_4, l_5\}$. We assume that the set of service transfers between nodes has already been processed by n_i, leading to the relation graph illustrated in Figure 5.5.

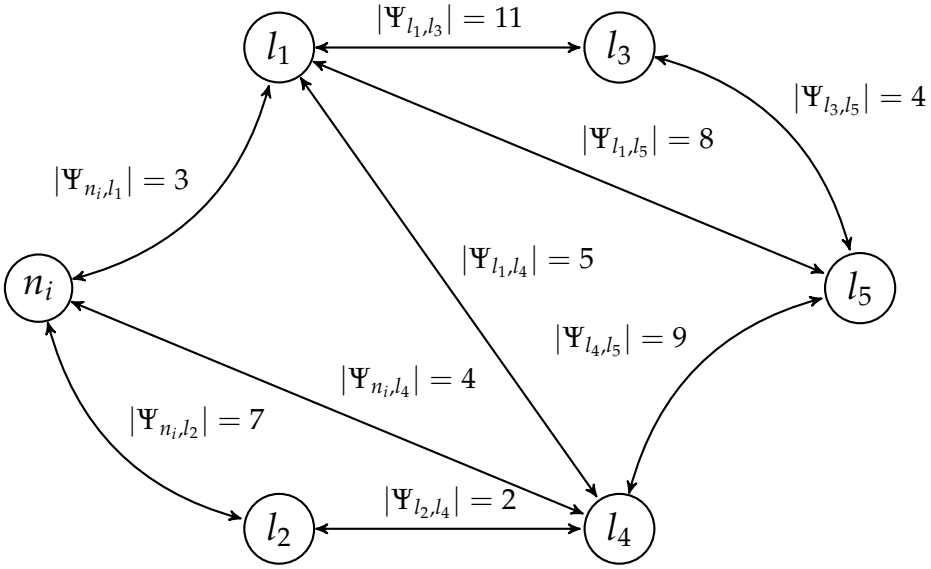


Figure 5.5: Relation graph of potential service transfers

Based on this information, the algorithm starts its first phase by calculating T_Ψ. So phase one ends with the table of matrix presented in Figure 5.6. In the second phase, we need to define for each node its best partner that contributes to maximize the service transfers in the whole system. In the iteration $loop_1$ the pair (l_1, l_3) is identified first. This is because (l_1, l_3) returns the maximum number of service transfers in T_Ψ. Eliminating them gives $X = \{n_i, l_2, l_4, l_5\}$. Next, pair (l_4, l_5) is identified in $loop_2$ and its elimination yields $X = \{n_i, l_2\}$. Finally, the pair (n_i, l_2) is identified and its elimination gives $X = \{\emptyset\}$. Hence, the algorithm finishes with the following optimization pairs $\{(l_1, l_3), (l_4, l_5), (n_i, l_2)\}$.

$T_\Psi =$

	n_i	l_1	l_2	l_3	l_4	l_5
n_i		3	7	0	4	0
l_1			0	11	5	8
l_2				0	2	0
l_3					0	4
l_4						9
l_5						

Figure 5.6: A simplified representation of T_Ψ after the execution of phase one

5.7 Evaluation

In this section an evaluation for the introduced self-optimization approach is provided. For the purpose of evaluating and testing, an evaluator based on the TEM middleware [ASM+13] has been implemented which is able to simulate the self-optimization algorithm. The evaluation network consists of 100 nodes, where all nodes are able to communicate with each other using message passing. Experiments with more nodes were tested and yielded similar results, but with 100 nodes more observable effects were seen. Each node has a limited resource capacity (memory) and is judged by an individual trust value without any central knowledge. Furthermore, four type of nodes are defined with different trust and resource values (see Table 5.1).

Table 5.1: For reasons of consistency with the previous self-configuration results, the same network setting is considered as in Table 4.1.

Node Type	Memory (MB)	Trust	Amount (%)
Type 1	[500 - 1000]	[0.7 - 0.9]	10
Type 2	[500 - 1500]	[0.3 - 0.6]	50
Type 3	[2000 - 4000]	[0.4 - 0.8]	30
Type 4	[4000 - 8000]	[0.4 - 0.9]	10

Then, a mixture of heterogeneous services with different resource consumptions are randomly generated for nodes. The sum of all node's service consumptions does not exceed a node's capacity (i.e., as defined in Formula 5.1). If, for exam-

ple, a trustworthy node is already full, then the same procedure is repeated for an untrustworthy node and so on until the average load of the system reaches 50% ($\overline{workload} = 50\%$). This means that some nodes may have many services and others none to unbalance the workload between nodes. Important services are created only for untrustworthy nodes and unimportant services for trustworthy nodes. Without the self-optimization techniques the workload of nodes are still unbalanced. Moreover, important services running on untrustworthy nodes are prone to fail. With the use of direct trust and reputation, the trust of a node can be measured and taken into consideration for the transfer of services. Two rating functions are used to evaluate the fitness of a service distribution regarding trust and workload. The first rating function for workload $\mathcal{F}_{workload}$ aims to calculate the average deviation of all nodes from the desired workload $\overline{workload}$ (in our case, 50%). This is expressed by the Formula 5.9, where $\mathcal{N}$ is the set of all nodes and $|\mathcal{N}|$ the cardinality of $\mathcal{N}$. The main idea of the second rating function $\mathcal{F}_{trust}$ is to

$$\mathcal{F}_{workload} = \frac{\sum_{n \in \mathcal{N}} |workload(n) - \overline{workload}|}{|\mathcal{N}|} \tag{5.9}$$

$$\overline{workload} = \frac{\sum_{n \in \mathcal{N}} workload(n)}{|\mathcal{N}|} \tag{5.10}$$

reward important services running on trustworthy nodes. This is expressed by the Formula 5.11, where $\mathcal{N}$ is the set of all nodes, $\mathcal{S}_n$ is the set of services on a node n, $t(n)$ its trust value and p(s) the priority of a service s (i.e., if s is important, $P(s)$ has the value of 1, otherwise 0).

$$\mathcal{F}_{trust} = \sum_{n \in \mathcal{N}} \sum_{s \in \mathcal{S}_n} p(s)t(n) \tag{5.11}$$

At the beginning of the simulation, the network is rated by using both $\mathcal{F}_{trust}$ and $\mathcal{F}_{workload}$. Then, the simulation is started and after each optimization step the network is rated again. Within one optimization step, 50 pairs of nodes (sender/receiver) are randomly chosen to perform the self-optimization process, i.e., $\rho = 50\%$. Senders send an application message to receivers to piggyback necessary information for the self-optimization, as described in section 5.3. Based on the extracted information the receiver determines whether it transfers its services or not. The goal is to maximize the availability of important services, which means that $\mathcal{F}_{trust}$ should be maximized (i.e., to an optimal theoretical point that we explain later

in 5.7.2). Therefore, it is necessary to transfer the more important services to more trustworthy nodes. Furthermore, the overall utilization of resources in the network should be well-balanced, i.e., $\mathcal{F}_{workload}$ should be minimized near to zero.

5.7.1 Results regarding the rating function $\mathcal{F}_{workload}$

As mentioned above, the first rating function $\mathcal{F}_{workload}$ indicates the average workload deviation of all nodes from the desired workload $\overline{workload}$ (in our case, 50%). The lower the value of $\mathcal{F}_{workload}$, the better the performance of workload balancing.

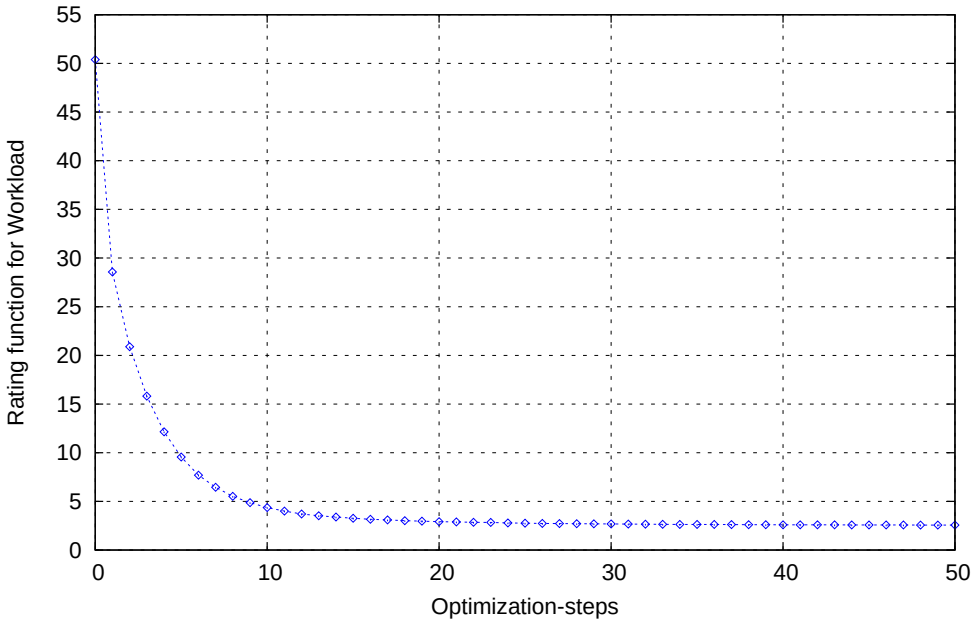


Figure 5.7: Rating function for the workload deviation ($\mathcal{F}_{workload}$)

Figure 5.7 shows the result of this experiment, whereas the values on the x-axis stand for optimization steps and the average workload deviation of nodes is depicted on the y-axis. It can be observed that the proposed algorithm improves the workload balancing by about 93%. However, it does not reach the theoretical maximum rate of 100% due to the trade-off between trust and workload.

5.7.2 Results regarding the rating function $\mathcal{F}_{trust}$

In the following, the service distribution for the proposed self-optimization algorithm is evaluated regarding $\mathcal{F}_{trust}$.

Figure 5.8 shows the result of this experiment. The square line represents the result of $\mathcal{F}_{trust}$ using the proposed self-optimization algorithm. It can be observed that the algorithm improves during runtime the availability of important services.

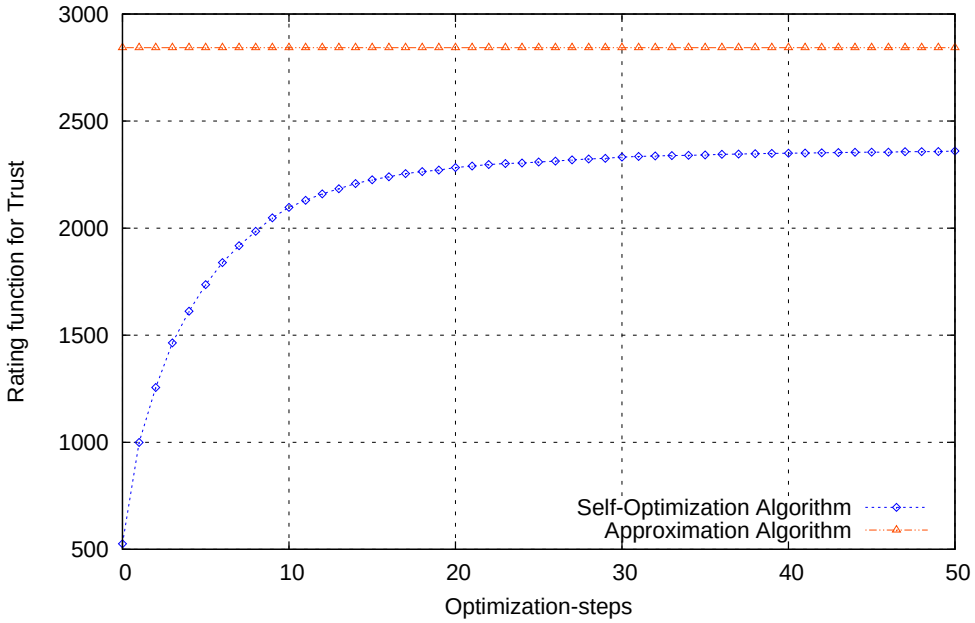


Figure 5.8: Rating function for Trust ($\mathcal{F}_{trust}$)

This means that the consideration of workload does not prevent the algorithm to relocate important services to trustworthy nodes. However, it remains to investigate the quality of the obtained result compared to an optimal theoretical result, when all important services are hosted only on trustworthy nodes (pure trust distribution, i.e., regardless of whether nodes are balanced or not). For this purpose we use an approximation algorithm that sorts in decreasing order the trust values of nodes and relocates all important services only to most trustworthy nodes until their capacity is full. The triangular marked line in the figure illustrates the result of the approximation algorithm. As a conclusion to all simulations we have done so far (about 1000 runs were evaluated) we can state that the proposed algorithm greatly improves the trust distribution of services. More precisely, it achieves 85% of the theoretical maximum result. However, it stays by 15% behind the theoretical maximum result due to the trade-off between trust and workload.

5.7.3 Basic Algorithm vs. Extensions

In this section, the gain of applying the proposed extensions with respect to Section 5.6 is investigated. We use the similar parameter settings of the initial evaluation, but we now allow for a certain percentage of randomly chosen nodes to receive multiple optimization requests simultaneously. This has the benefit to put the evaluation more in a context of real life. In this part of work, the following three algorithms are compared regarding their ability to perform the optimization in the

system.

- **Basic algorithm (ALG.1):** The basic optimization algorithm as in the previous experiments.
- **Basic algorithm + Selective Request Handling (ALG.2):** A variation of the basic optimization algorithm using the extension of the selective request handling (see Section 5.6.1).
- **Basic algorithm + Parallel Request Handling (ALG.3):** A variation of the basic optimization algorithm using the extension of the parallel request handling (see Section 5.6.2)

The three algorithms differ in the way they handle multiple requests, either sequential or parallel. Figures 5.9 and 5.10 present their comparison results with respect to the rating functions $\mathcal{F}_{trust}$ and $\mathcal{F}_{workload}$.

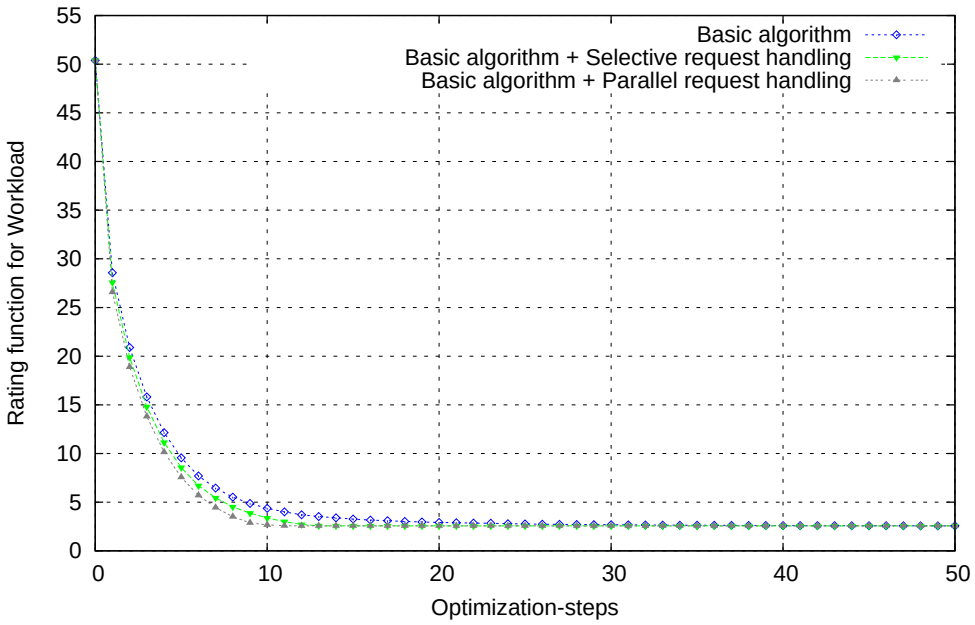


Figure 5.9: Comparison results according to the rating function $\mathcal{F}_{workload}$

It is easy to see that both investigated variations of ALG.2 and ALG.3 indeed provide an even better optimization time than the basic algorithm ALG.1, especially the variation of ALG.3, currently shows the best time performance to achieve the optimization process. This is due to its ability to support parallelism through the optimization of requesters such that every one optimizes itself with the node with the highest gain of service transfers. This results — in the whole system — to a reduce of the processing time into the overall optimization.

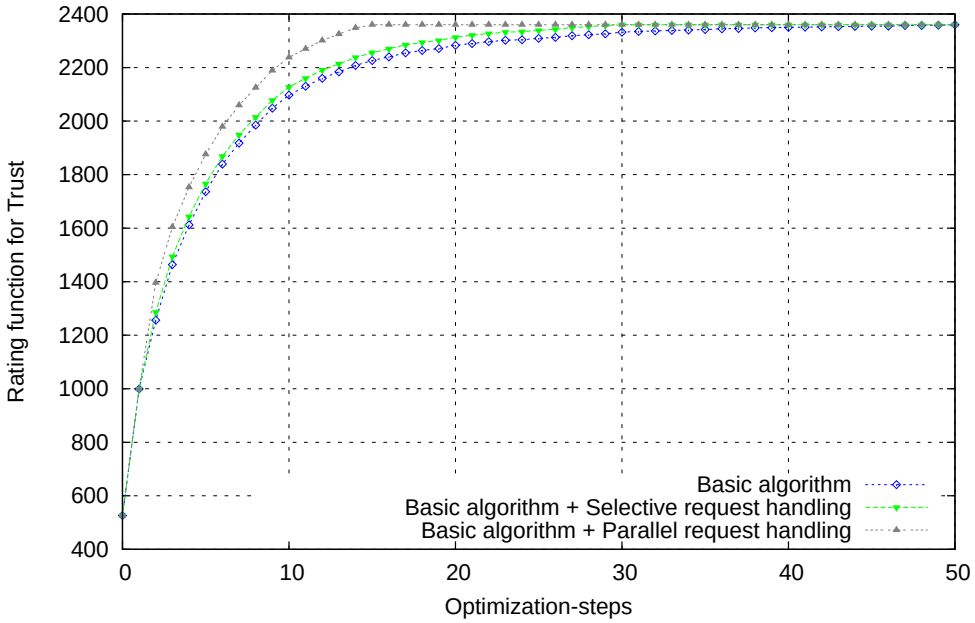


Figure 5.10: Comparison results according to the rating function $\mathcal{F}_{trust}$

5.7.4 Different Network Settings

In the following, additional experiments are conducted to further investigate the behavior of the introduced self-optimization algorithm with different network settings. We performed a binary classification of nodes with a ratio of 50/50, and for each classification type, we generated a different amount of memory resources and trust values, as shown in Table 5.2. Generally, the more trustworthy the nodes are,

Table 5.2: A binary classification of heterogeneous nodes

Node Type	Memory (MB)	Trust	Amount (%)
Type 1	[8000 - 16000]	[0.6 - 0.99]	50
Type 2	[1000 - 8000]	[0.1 - 0.60]	50

the higher is the amount of their memory resources. We argue that this is a useful and realistic network parametrization since it enables to model the behaviour of servers and workstations which are expected to be trustworthy in real-world situations through the use of Type 1 as well the behavior of mobile devices (i.e., expected in real-world to be less trustworthy than servers and workstations) through the use of Type 2. The average workload $\overline{workload}$ is set to 45%. The experiments differ in the adjustment of $|\mathcal{N}|$ and ρ. Recall, $|\mathcal{N}|$ states for the size of the network and ρ rep-

resents the percentage amount of involved nodes within one optimization step to perform the optimization process. In the following the results of conducted experiments are presented. To ensure representative values, any experiment is repeated 300 times and the results are averaged.

The first three experiments examine the behaviour of the self-optimization algorithm with a fixed $|\mathcal{N}| = 100$ but different percentage of ρ.

- Experiment 1.1: $|\mathcal{N}| = 100, \quad \rho = 30\%$ (see Figures 5.11 and 5.12)
- Experiment 1.2: $|\mathcal{N}| = 100, \quad \rho = 50\%$ (see Figures 5.11 and 5.12)
- Experiment 1.3: $|\mathcal{N}| = 100, \quad \rho = 70\%$ (see Figures 5.11 and 5.12)

Experiments 2.1-2.3 consider a fixed network size of $|\mathcal{N}| = 200$ and different percentage of ρ.

- Experiment 2.1: $|\mathcal{N}| = 200, \quad \rho = 30\%$ (see Figures 5.13 and 5.14)
- Experiment 2.2: $|\mathcal{N}| = 200, \quad \rho = 50\%$ (see Figures 5.13 and 5.14)
- Experiment 2.3: $|\mathcal{N}| = 200, \quad \rho = 70\%$ (see Figures 5.13 and 5.14)

The following three experiments are similar to the first ones but the the network size is set to $|\mathcal{N}| = 400$.

- Experiment 3.1: $|\mathcal{N}| = 400, \quad \rho = 30\%$ (see Figures 5.15 and 5.16)
- Experiment 3.2: $|\mathcal{N}| = 400, \quad \rho = 50\%$ (see Figures 5.15 and 5.16)
- Experiment 3.3: $|\mathcal{N}| = 400, \quad \rho = 70\%$ (see Figures 5.15 and 5.16)

The last three experiments examine the behaviour of the introduced algorithm with $|\mathcal{N}| = 800$ and different ρ.

- Experiment 4.1: $|\mathcal{N}| = 800, \quad \rho = 30\%$ (see Figures 5.17 and 5.18)
- Experiment 4.2: $|\mathcal{N}| = 800, \quad \rho = 50\%$ (see Figures 5.17 and 5.18)
- Experiment 4.3: $|\mathcal{N}| = 800, \quad \rho = 70\%$ (see Figures 5.17 and 5.18)

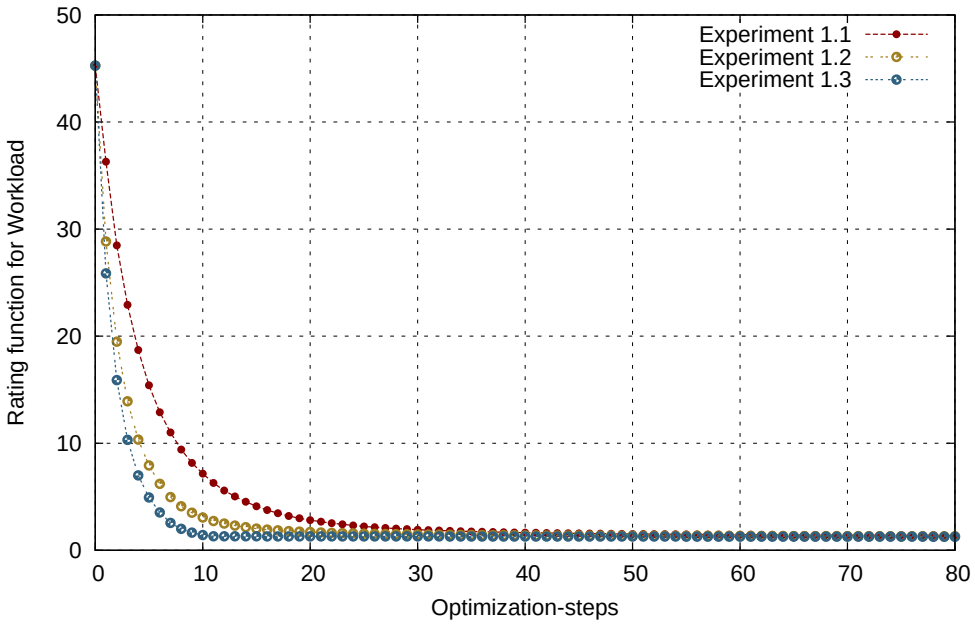


Figure 5.11: Result of experiments 1.1 - 1-3 according to the rating function $\mathcal{F}_{workload}$

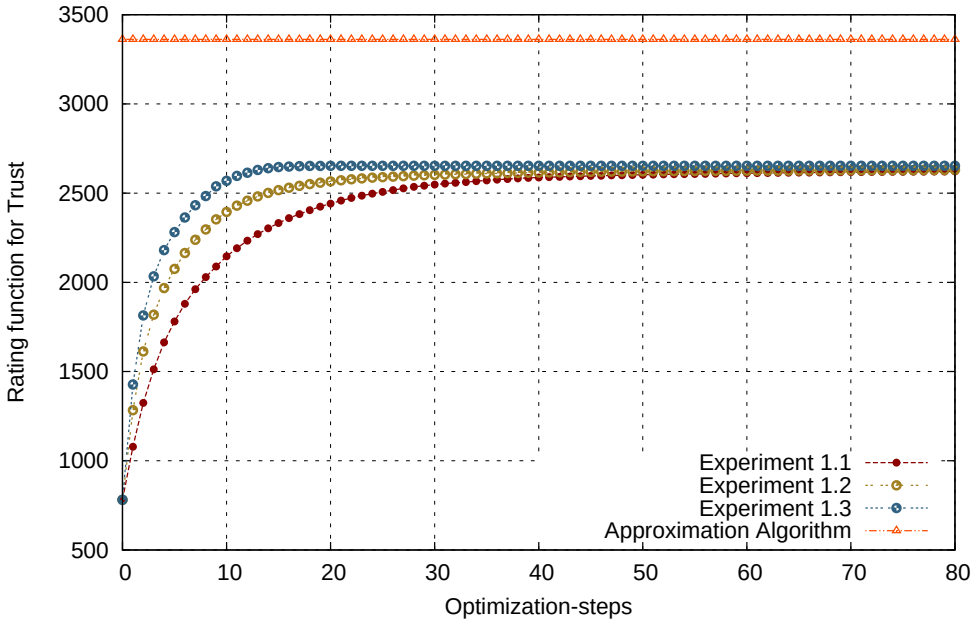


Figure 5.12: Result of experiments 1.1 - 1-3 according to the rating function $\mathcal{F}_{trust}$

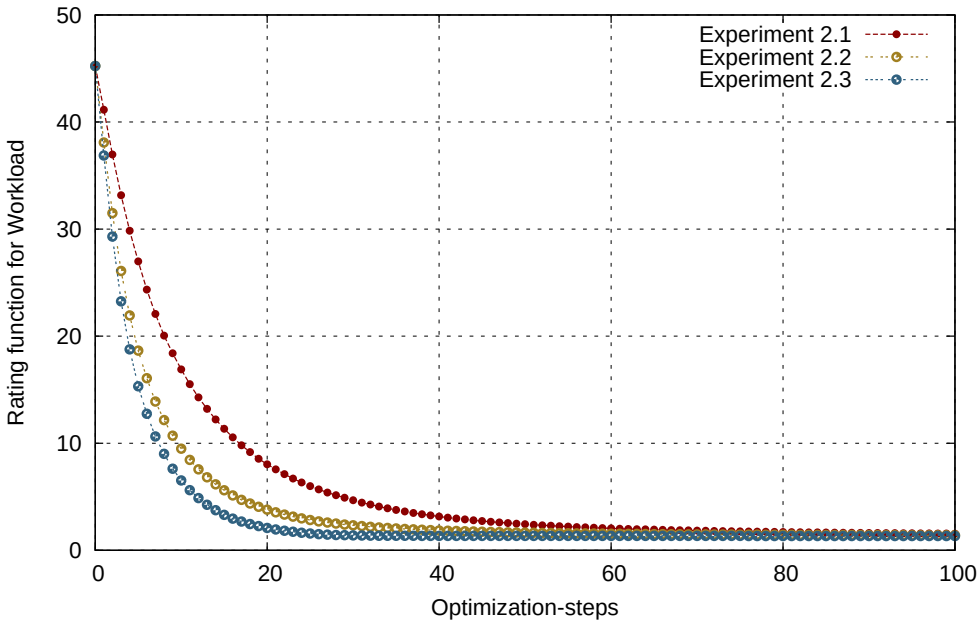


Figure 5.13: Result of experiments 2.1 - 2-3 according to the rating function $\mathcal{F}_{workload}$

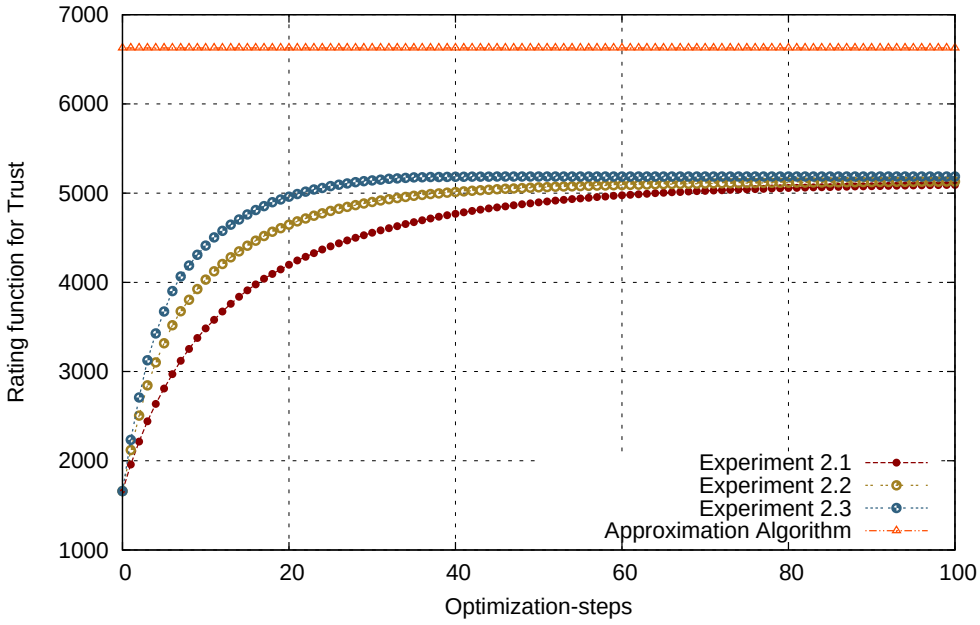


Figure 5.14: Result of experiments 2.1 - 2-3 according to the rating function $\mathcal{F}_{trust}$

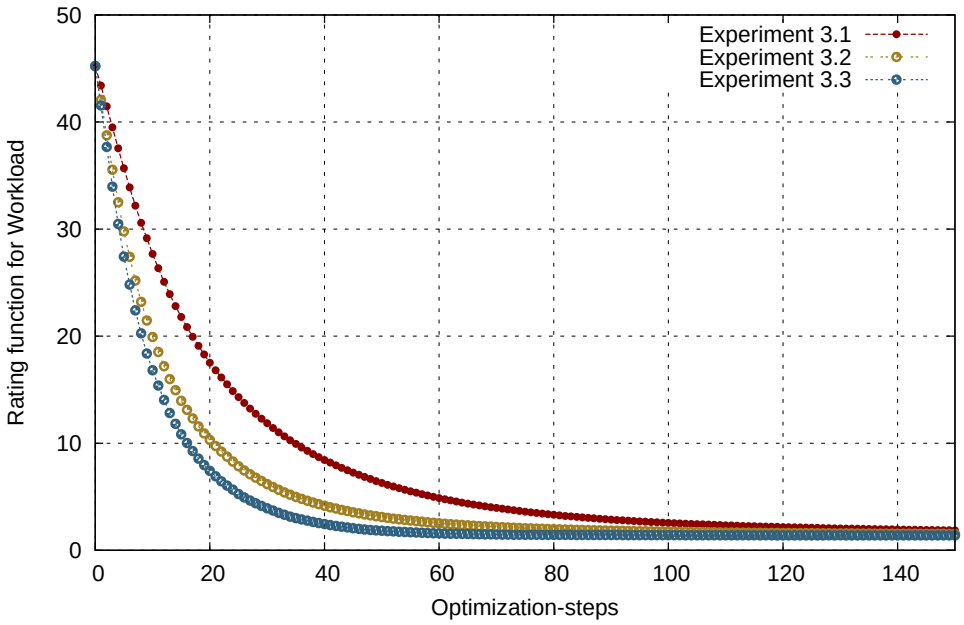


Figure 5.15: Result of experiments 3.1 - 3-3 according to the rating function $\mathcal{F}_{workload}$

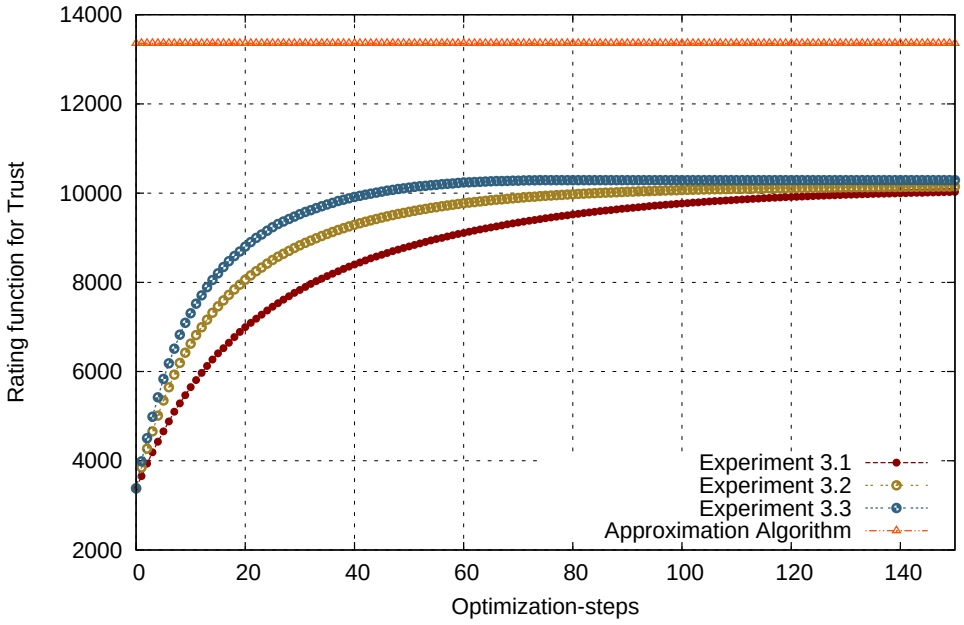


Figure 5.16: Result of experiments 3.1 - 3-3 according to the rating function $\mathcal{F}_{trust}$

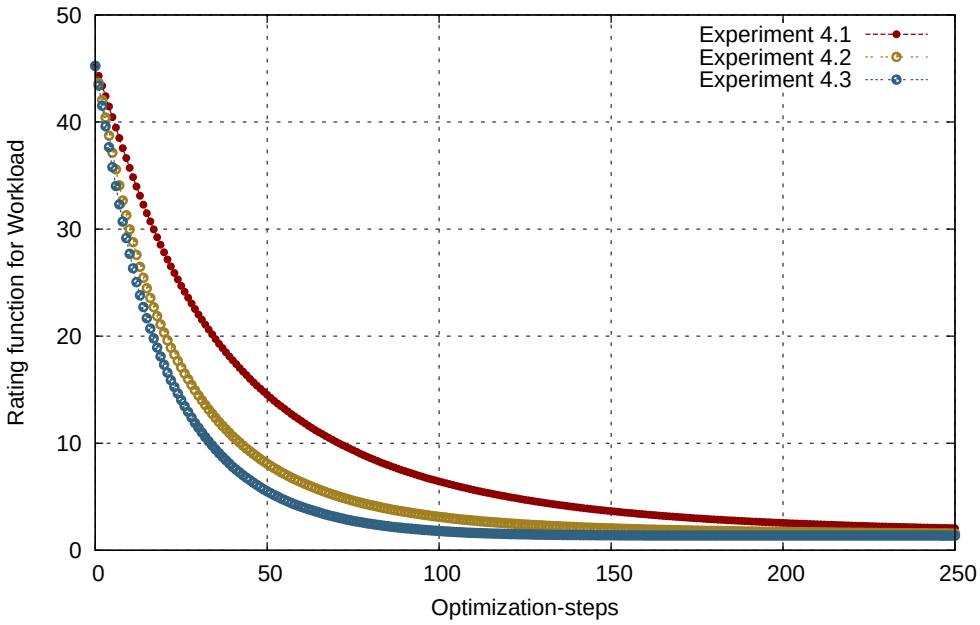


Figure 5.17: Result of experiments 4.1 - 4-3 according to the rating function $\mathcal{F}_{workload}$

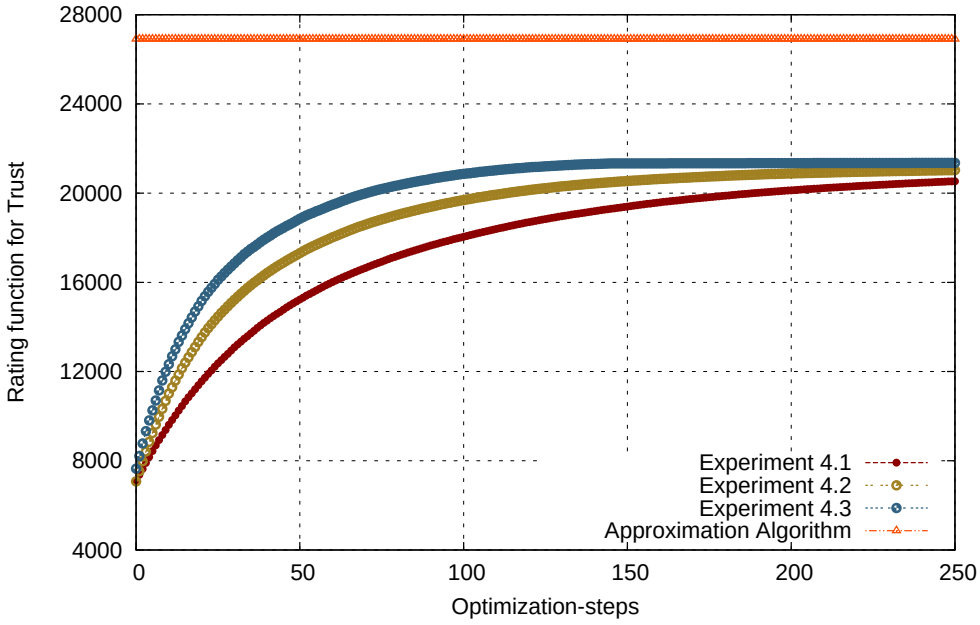


Figure 5.18: Result of experiments 4.1 - 4-3 according to the rating function $\mathcal{F}_{trust}$

Conclusion Deduced From Conducting Experiments. The experiment results, with the focus on workload, are depicted in Figures 5.11, 5.13, 5.15, and 5.17. These figures show the optimization steps on the horizontal axis and the workload deviation of nodes on the vertical axis. Values near to the bottom left corner represent small deviation of workloads with few number of optimization steps. The results attest the introduced self-optimization algorithm a continuous reduction of the workload deviations in all kind of settings. Beside the workload balancing, the introduced algorithm provides also a good ability to improve its speedup over the parametrization of ρ, making it suitable to be applied in overfilled situations with too many number of messages. Figures 5.12, 5.14, 5.16, and 5.18 show similar results to the workload experiments, but with the focus on trust. The optimization steps are depicted on the horizontal axis and the fitness function for trust on the vertical axis. Optimal theoretical values considering pure trust distributions are marked with red triangular lines for each experiment. Similarly to the last results, we can state that the algorithm developed in this work is able to always improve the availability of important services at runtime and that the parametrization of ρ plays here also an important role to increase the speedup of the trust optimization in the whole system.

5.8 Conclusions and Future Work

In this chapter, a novel self-optimization algorithm for open distributed self-* systems has been proposed. The algorithm does not only consider pure load-balancing but also takes into account trust to improve the assignment of important services to trustworthy nodes at runtime. More precisely, the algorithm makes use of different optimization strategies —- as cited in the corresponding part of Section 5.5 — to determine whether a service should be transferred to another node or not. Section 5.7 presents the results of the performance measurements that are conducted to evaluate the algorithm. The results indicate that for our model trust concepts improve significantly the availability of important services while causing a small deterioration (i.e., by about 7%) regarding load balancing. Therefore, we classify our algorithm as a kind of best-effort approach that provides good but not necessarily optimal solutions to this trade-off problem. Then, a set of variations of the basic algorithm are introduced in Section 5.6 to improve its performance in case of multiple requests. The difference between the variations arises in the way to handle requests, either sequential or parallel. In Section 5.7.3, a comparative evaluation is conducted to analyze the performance results of the variations compared to the basic approach. The results attest a good performance for the extended optimization algorithm with parallel request handling. In Section 5.7.4, an additional evalua-

tion is provided to further investigate the behavior of our approach for different network settings. The results indicate here as well a good performance for our algorithm. It clearly attains its goals of both trust and load optimizations in all kind of parametrizations and network sizes. Apart form this, the algorithm provides also a good possibility to increase its speedup over the parametrization of ρ making it suitable to be applied in overfilled situations with too many number of messages. In future work, extensions are planed to deal with the Cold-Start-Problem, i.e., the need to integrate new nodes with unknown trust values with other nodes in the network. This is very important to improve the robustness of the proposed self-optimization algorithm. One possible solution to address this issue could be to make runtime prediction or online training for the new participating nodes, but as it goes beyond the scope of this work it is not further discussed here.

6
Conflicting Trust Values

Abstract. In an open self-* system with nodes representing machines, the trust behavior of node participants may change during runtime. Every node in the system can execute services of other nodes and also provide services to other nodes. It has to decide using trust for which nodes it wants to execute the services and to which nodes it wants to give its services. This service distribution is performed by using our former developed trust-based self-configuration approach, which were devised to improve the availability of important services. However, this works only well for environments, where all nodes see the same trust value for a certain node. Hence, the baseline algorithm is not suited to operate with conflicting trust values. This situation can occur by collecting trust values independently from the neighbors of a node that contradict each other. The contribution of this chapter is a conflict resolution mechanism as an extension to the baseline algorithm to operate with conflicting trust values at the same time.

6.1 Introduction

Current self-* systems — such as AC/OC — focus on developing novel mechanisms capable of so-called self-* properties to manage the growing complexity of today's computing systems. In such systems, trust has become an important aspect to reduce the information uncertainty of misbehaving nodes. With appropriate

trust techniques, nodes in the system can have a clue about which nodes to cooperate with. This is very important to improve the robustness of self-* systems, which depends on a cooperation of autonomous nodes. So far, we have enhanced the self-configuration and self-optimization properties with trust capabilities (for more details we refer to Chapters 4 and 5) but these approaches were based on the assumption that nodes in the system have the same trust value in a certain node. This means that our former work on the self-* properties is not suited to operate with conflicting trust values at the same time. This situation can happen by collecting trust values independently from the neighbors of a node that can contradict each other. The contribution of this chapter is a conflict resolution mechanism as an extension to our former work to cope with the problem of conflicting trust values in self-* systems. The chapter offers as contribution the following aspects:

(i) A highlight of our former developed work to easily put the reader in the context needed for understanding the problem of conflicting trust (see Section 6.2),

(ii) a description of the recognized problem using a simple example to indicate the specific purpose of our research (see Section 6.3),

(iii) an adaptive opinion mechanism enabling nodes in the system to individually determine their opinion values at runtime (see Section 6.4), and

(iv) a conflict resolution mechanism — as an extension to our previous works — to cope with the problem of conflicting trust values (see Section 6.5).

All aspects are evaluated and discussed with respect to a toolkit based on the TEM [ASM+13], a trust-enabling middleware for building real-world distributed Organic Computing systems. Section 6.6 provides evaluation results of the proposed mechanism. Finally, the chapter is closed with a conclusion and future work in Section 6.7.

6.2 Previous and Related Work

Previous Work. Recall, in this thesis we adopt the same definition of trust of the OC-Trust research unit (see Chapter 2.4). We see that trust consists of several facets, as for example, safety, reliability, credibility and usability. Our investigation focuses in this part of work on the reliability aspect. Furthermore, it is assumed that a node can not realistically assess its own trust value because it trusts itself fully. Therefore, the calculation of the trust values in this thesis is done with the trust metrics introduced in Chapter 3.4.2 that can be summarized as follows:

- **Direct Trust** is based on the own experiences a node has made directly with an interaction partner node. Typically, trust values are calculated by taking the mean or weighted mean of past experiences.
- **Reputation** is based on the trust values of others that had experiences with the interaction partner. Reputation is typically collected if not enough or outdated own experiences exist.
- **Confidence** Before both values, direct trust and reputation, can be aggregated to a total trust value, the reliability of one's own trust value has to be determined, the so-called confidence. If a node does have a direct trust value but is not confident about its accuracy, it needs to include reputation data as well.

When all the aforementioned values are obtained, a total trust value t_{n_i,n_j} based on direct trust and reputation values can be calculated using confidence to weight both parts against each other (see Chapter 3). This value represents the current trust of n_i in node n_j and will always range between 0 and 1. The value of 0 means that n_i does not trust n_j at all while a value of 1 stands for complete trust. These trust metrics are important to enhance the self-* properties of self-* systems with trust capabilities (i.e., the self-configuration introduced in Chapter 4 and the self-optimization provided in Chapter 5). However, these trust-enhanced properties perform only well when no contradicting trust values occurs. Therefore, we think that there is room for further improvement with the mechanism presented in this chapter.

Related Work. The presented conflict resolution algorithm differs from the most existing state of the art manager election mechanisms, e.g., [BA06, LB11] in two important points: First, it takes into account the trust constraints of nodes to elect managers. Thus, our algorithm can be applied to untrustworthy networks since trustworthy and untrustworthy nodes are treated differently. Furthermore there is no global instance that coordinates all managers. Therefore, we want to compare it to systems employing the same approach. A very similar system can be found in [VCPG06], where some efforts have been done to select leaders based on their reasonable trust levels. The system achieves interesting results in preventing the election of untrustworthy leaders. However, a main disadvantage is that it is not suitable to deal with colluding nodes, as we do in this work. Parts of the content of this chapter have been published by the author in the following conference and book chapter:

- **[MSKU15]:** Nizar Msadek, Alex Stegmeier, Rolf Kiefhaber, and Theo Ungerer *A Mechanism for Minimizing Trust Conflicts in Organic Computing systems*.

In SAOS 2015: Proceedings of the second International Workshop on Self-Optimisation in Organic and Autonomic Computing Systems in conjunction with ARCS 2015, pages 1-7, Porto, Portugal, IEEE Computer Society, 2015.

6.3 The Problem of Conflicting Trust Values

Our application scenario is a real organic computing system with nodes representing client machines, which can interact with each other through a set of messages. The system is distributed without a global control. The considered applications are composed of services, which are distributed among the network. Such a scenario would be suited for systems where clients run applications that produce large amount of services and thus are in high demand of computing resources, e.g., face recognition [Rob09] or ray tracing [Gla89]. We distinguish two types of nodes: *managers* and *contractors*. A *manager* is responsible for hosting services and collecting results, whereas a *contractor* is responsible for the execution of services. In our scenario, contractors have to cope with untrustworthy managers, which might be unavailable for large periods of time. This means that their services and previously computed results might be unavailable as well. This situation would harm the efficiency of the overall system as it leads to repeated execution of services and resubmission of results. By introducing trust, contractors can identify those untrustworthy managers prior to fail and move their services to more trustworthy managers. This issue was addressed in a former work of trust-enhanced self-optimization and has the drawback that it works only well under the assumption that all nodes have the same trust value in a certain node. However in a real life situation, contractors should be able to operate with conflicting trust values. These conflicts are caused by collecting trust values independently from the neighbors of a node that can contradict each other. Figure 6.1 visualizes this problem in a short example of three nodes.

Considering a network with just three nodes: two contractors $c1$ and $c2$ and one managers $m1$. Let us now suppose that a shielding wall is set between two nodes i.e., c_2 and m_1, producing poor reliability values between them, while the third node c_1 is not affected. In this case, contractor c_2 considers the manager m_1 as untrustworthy and thus not able to properly host services. Hence, it wants to nominate another manager for taking over the current manager's role, while contractor c_1 sees no need for action. Such situations cause conflict between contractors and must be resolved. Therefore, we analyze in this chapter the problem of conflicting trust values and propose a solution as extension to our former work.

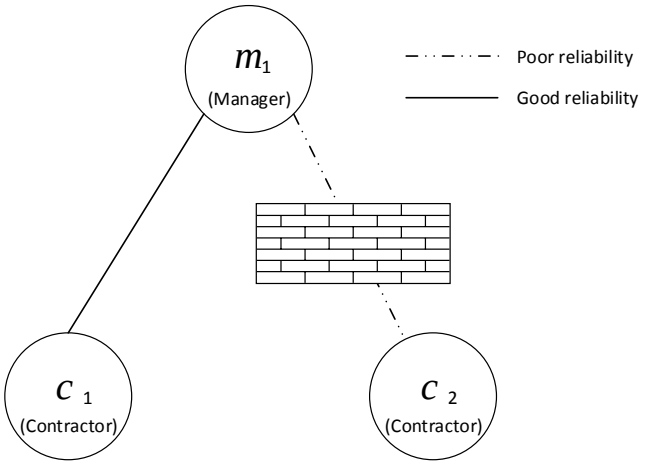


Figure 6.1: The conflicting trust values problem in an example of three nodes

6.4 Metric for Opinion

In this section a metric is given to enable contractors to calculate individual opinion values about managers. This calculation plays an important role in the conflict resolution algorithm as it directly influences the overall satisfaction of contractors and thus contributes to reduce the number of conflicts between them. The metric reflects the subjective belief of contractor c in the ability of manager m to properly act as manager and is defined as follows:

$$opinion^{\overrightarrow{c,m}}(t_{c,m}) = \sqrt[q]{1 - (t_{c,m})^q} \tag{6.1}$$

where $t_{c,m}$ means the trust value a contractor c has about a manager m, i.e., this values is calculated based on our former work (see Section 6.2 for more details) and q represents the subjective behavior of c, since contractors can behave differently towards managers. The opinion value will always range between 0 and 1. The value of 1 means that manager m from the perspective of c is totally unqualified for hosting services and therefore it has to give off its manager role and transfer its services to another manager. However the value of 0 would represent a perfectly qualification for acting as manager. The border of changing the opinion from one belief to the other is located at 0.5. Figure 6.2 illustrates how the metric would look like using different values of q.

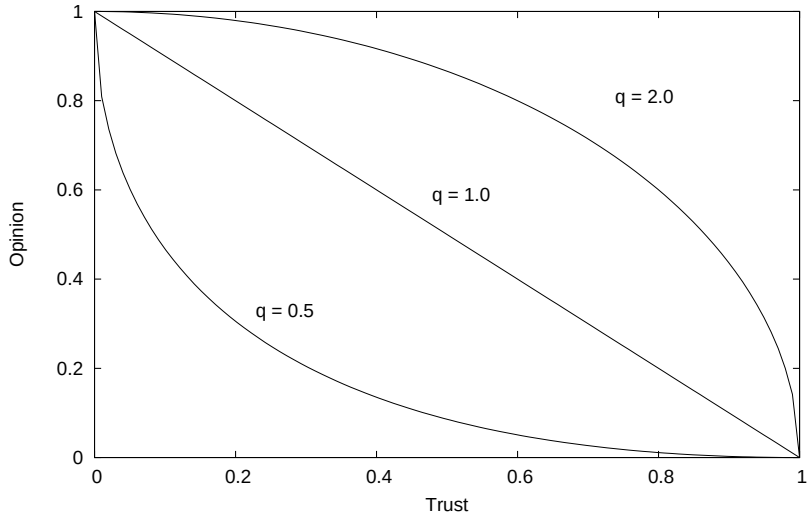


Figure 6.2: Graph representation of the opinion metric using different values of q

6.5 Conflict Resolution Mechanism

In this section a new trust conflict resolution mechanism that can be extended to our former work is presented. It is designed for flexible generic usage as a basis to reduce trust conflicts between contractors. Figure 6.3 provides a global outline of it as EPC diagram. The description of the algorithm is given in the following sections.

6.5.1 Extracting Opinions

In Figure 6.4, the *Extracting opinions* process is formalized as a sequence diagram. When a dissatisfied contractor detects a trust conflict, it notifies the existence of that conflict to its manager with an alert message. This message contains a list of candidate nodes that the dissatisfied contractor believes they are suitable for taking over the role of manager. These candidates are determined based on the opinion metric presented in Section 6.4. The manager performs then a lookup operation to find other contractors that are involved in the conflict. It asks them to provide their opinions about all received candidates. This is achieved by using either a limited broadcast or a multicast message. Contractors that receive such a message evaluate each candidate using the opinion metric. The most suitable ones are those, which can remove or reduce the conflict to an acceptable level. Afterwards, each contractor communicates its opinion to the manager. If all opinions are received, the *Nomination* phase starts (See Section 6.5.2).

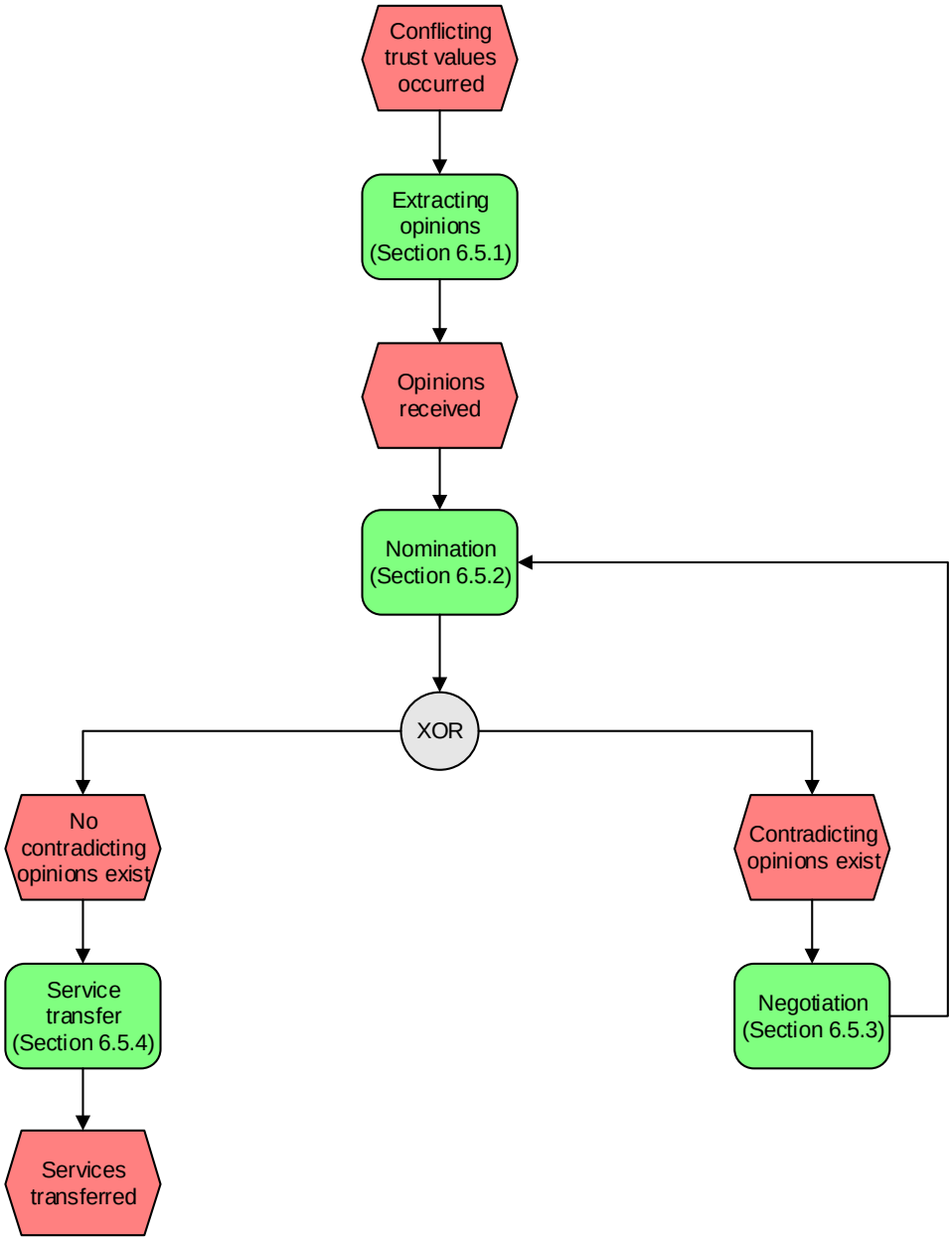


Figure 6.3: Event-driven process chain of the conflict resolution mechanism

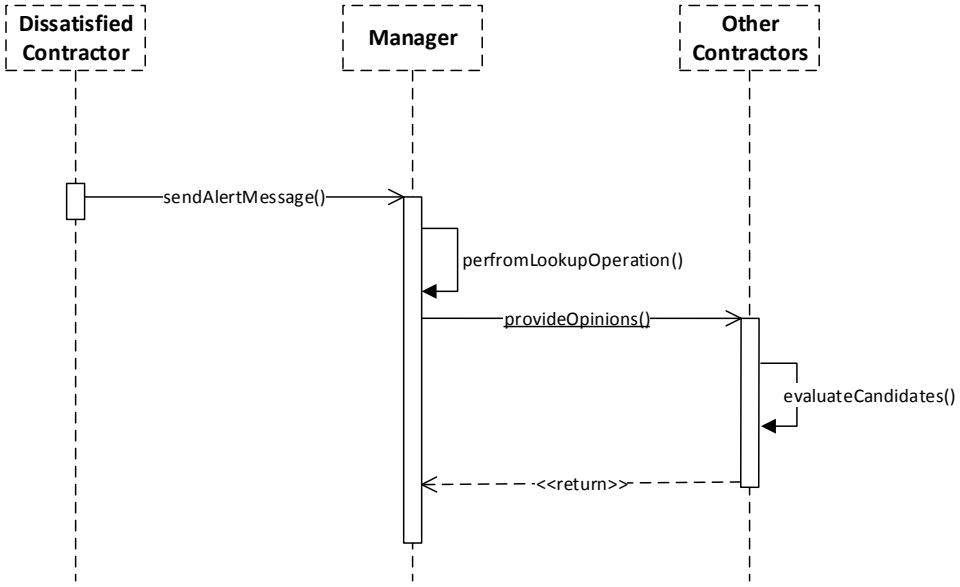


Figure 6.4: Sequence diagram for the Extracting opinions process

6.5.2 Nomination

The *Nomination* phase is illustrated in Figure 6.5. In this phase, the manager has to nominate one candidate for taking over its role as manager. Therefore, candidates are ranked such that good opinion of contractors (i.e., the closer the opinion values are to zero the more candidates are considered to be suitable) and low conflict number positively influence the rank. Note that the emphasis at that stage is more on the conflict number than on the opinion. The result of the best candidate will be then communicated to the contractors. It may happen that some contractors are not satisfied with the nominated candidate. In that case, they send a rejection message to the manager and trigger thereby the *Negotiation* process (See Section 6.5.3). Otherwise, the service transfer can take place between the manager and the nominated candidate (See Section 6.5.4).

6.5.3 Negotiation

After *Nomination*, several contractors could be dissatisfied with the nomination result. This might lead to inconvenience for the manager to decide who could take over its manager role. In that case as illustrated in Figure 6.6, the manager iterates through all dissatisfied contractors. Let be c the current dissatisfied contractor. It asks c to provide a list of new candidates, which should have not yet been proposed in earlier nominations and that c believes they are suitable to take over the role of manager. These candidates are submitted to the manager. The manager asks then

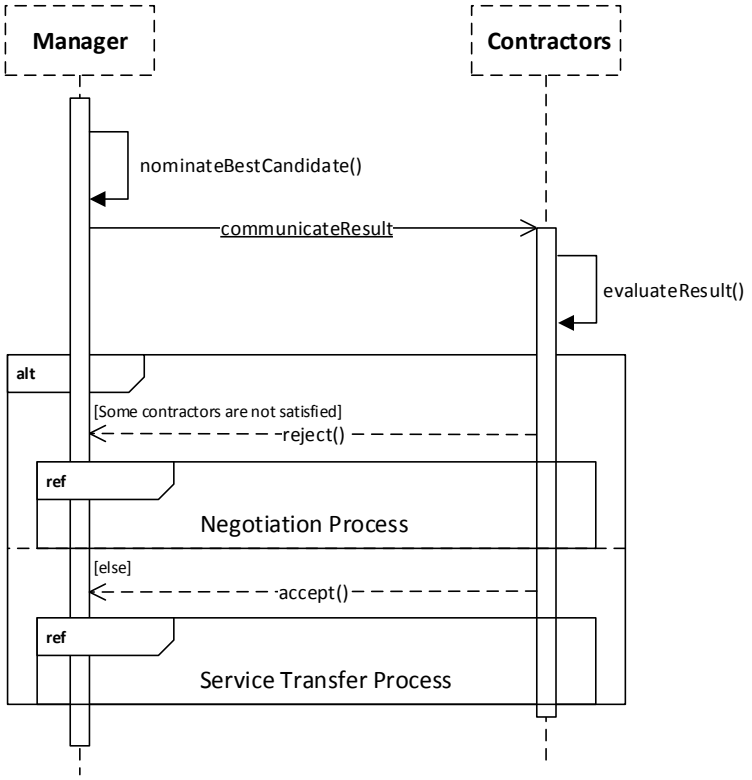


Figure 6.5: Sequence diagram for the Nomination process

the other contractors about their opinions about each one of them. All opinions are collected and finally the *Nomination* process (See Section 6.5.2) is performed again. If the *Nomination* phase succeeds, then the iteration breaks and the *Service transfer* between the manager and the nominated candidate takes place (See Section 6.5.4). Otherwise, the *Negotiation* process (See Section 6.5.3) is repeated until all dissatisfied contractors have been iterated. It may occur that at the end no candidate has been nominated because the manager could not find a candidate that other contractors were all satisfied with. In such case, the manager retains its manager role for a given time interval Δ_T before recalling the *Negotiation* process again, with the hope that in the future the result would be better, since it is assumed that contractors can change over time their opinions.

6.5.4 Service Transfer

After the *Nomination* process has finished without contradicting opinions, the service transfer takes place between the manager and the nominated candidate. The manager requests the contractors to remain idle and to stop using services during the period of service transfer. It saves the last computation states of all services on a reliable storage [TEP+07]. These computation states are referred as snapshots and

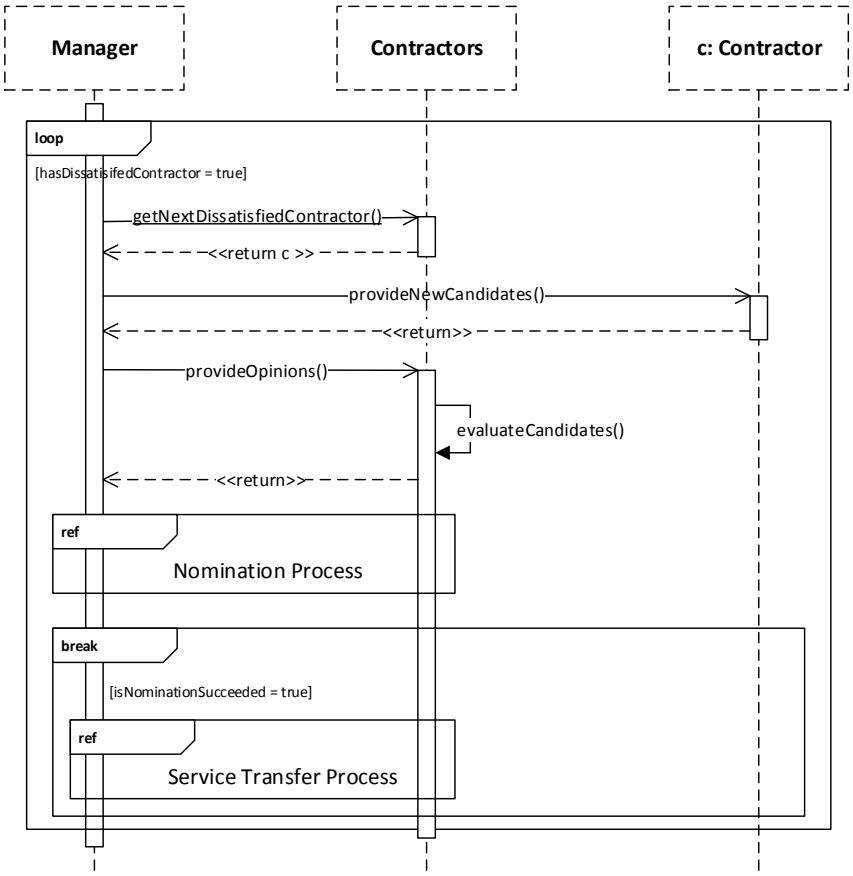


Figure 6.6: Sequence diagram for the Negotiation process

are used to allow the nominated candidate later to resume the forward execution of services without having to discard all the work done up to the time of service transfer. Afterwards, services are stopped and transferred together with their snapshots to the nominated candidate. Immediately after the service transfer, the manager gives off its manager role and the nominated candidate starts to act as manager. It uses the snapshots to restart the services from their last stored execution point, and finally, requests the contractors to resume their activities by using the services. In Figure 6.7, the above described process is formalized as a sequence diagram.

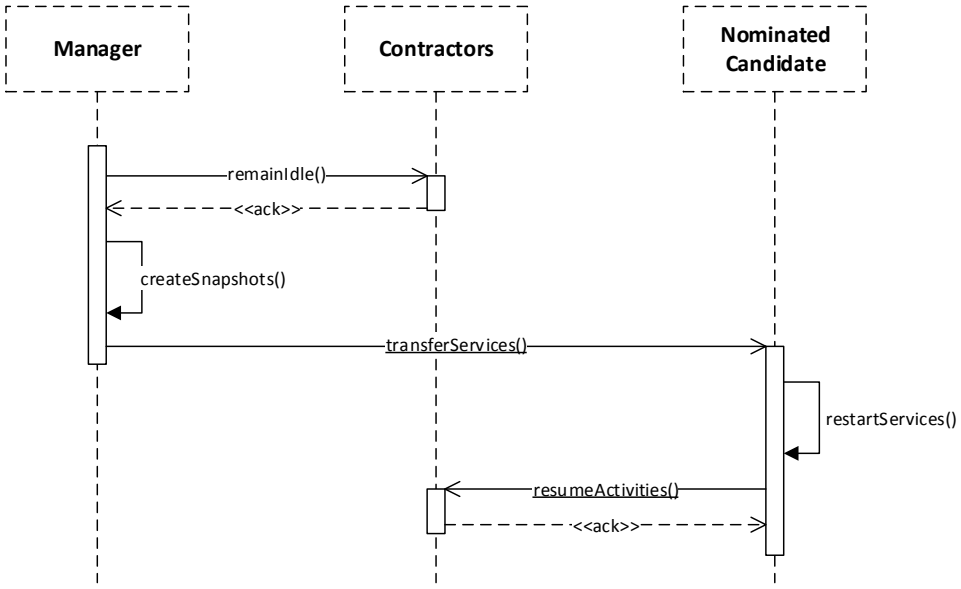


Figure 6.7: Sequence diagram for the Service transfer process

6.6 Evaluation

For evaluation purposes, an evaluator based on TEM [ASM+13] has been implemented which is able to simulate the introduced conflict resolution approach. It is written in Java and includes a conflict injection component enabling to simulate conflicts artificially between nodes. The evaluation network consists of 30 nodes, where all nodes are able to communicate with each other using message passing. Experiments with more nodes were tested and yielded similar results, but with 30 more observable effects were seen. Each node has a limited resource capacity (e.g., CPU and memory) and is judged by an individual trust value without any central knowledge. Then 10 services, 5 of them important and 5 unimportant are created with random values for each resource. The initial service assignment is performed randomly to ensures that the proposed approach is evaluated under a great vari-

ety of start conditions. After the assignment, each node adopts either the role of a manager, who possesses a service, or the role of a contractor, who executes a service hosted by a manager. Using the conflict injection component, different rates of conflict have been simulated. Our goal focuses on reducing those conflicts via the proposed algorithm. Additionally, it is measured how many messages are needed to accomplish the algorithm. Each evaluation scenario has been tested 500 times with randomly generated networks and the results are averaged. In the following the results of the conducted evaluations are presented.

6.6.1 Frequency of Conflicts Before and After Execution

As stated in the specification, the proposed conflict resolution algorithm is supposed to reduce the frequency of conflicts among contractors. This can be shown by comparing the total number of conflicts before and after its execution. Figure 6.3 shows the result of this experiment for different scenario cases. The Worst-case sce-

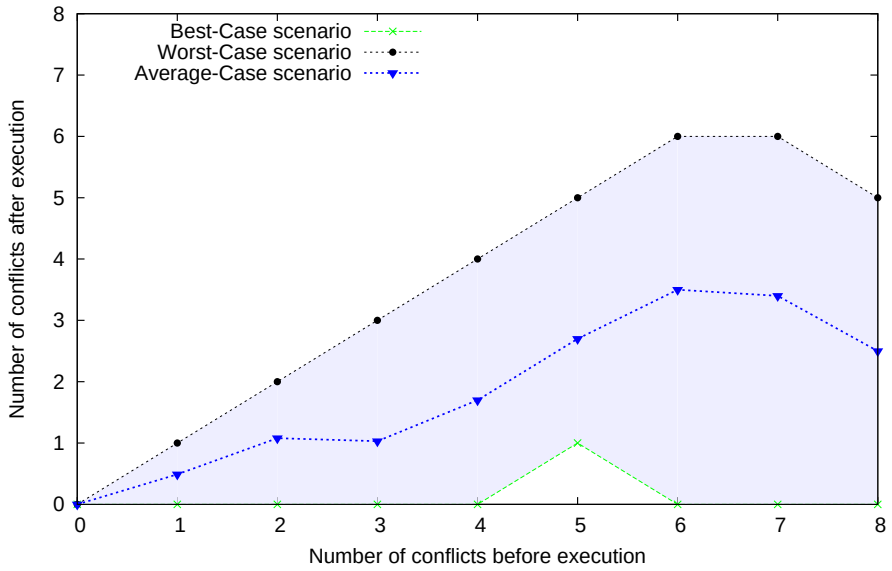


Figure 6.8: Frequency of conflicts before and after execution

nario represents the maximum number of observed conflicts, the Best-case scenario stands for the minimum number of observed conflicts and the Average-case scenario speaks for the averaged number of conflicts, i.e., in total, about 500 runs were evaluated. The evaluation results show that the proposed approach performs well for all investigated scenarios. It is able to reduce significantly the number of conflicts among contractors. To quantify the obtained enhancement of our approach, we calculate the average conflict reduction (ACR) as follows:

$$ACR = \frac{1}{n}\sum_{i=1}^{n}\frac{(C_i^{before} - C_i^{after})}{C_i^{before}} \tag{6.2}$$

where n means the number of conducted experiments (i.e., in our case $n = 8$). C_i^{before} represents the number of conflicts used for experiment i before the execution of our approach, whereas C_i^{after} stands for the resulting number of conflicts after the execution of the algorithm. As a conclusion to all simulations we have done so far (about 500 runs were evaluated) we can state that ACRs of 97.5%, 53,42% and 6,47% were achieved respectively to the Best-case, Average-case and Worst-case scenarios of the conflict resolution algorithm.

6.6.2 Mean Number of Messages

Regarding the amount of nodes and the number of messages needed to accomplish the conflict resolution algorithm the mean number of messages per node can be calculated. Figure 6.9 shows the result of this experiment, whereas the values on the x-axis stand for time steps and the mean number of messages is depicted on the y-axis. The red dashed lines illustrate the time at which conflicts between contractors

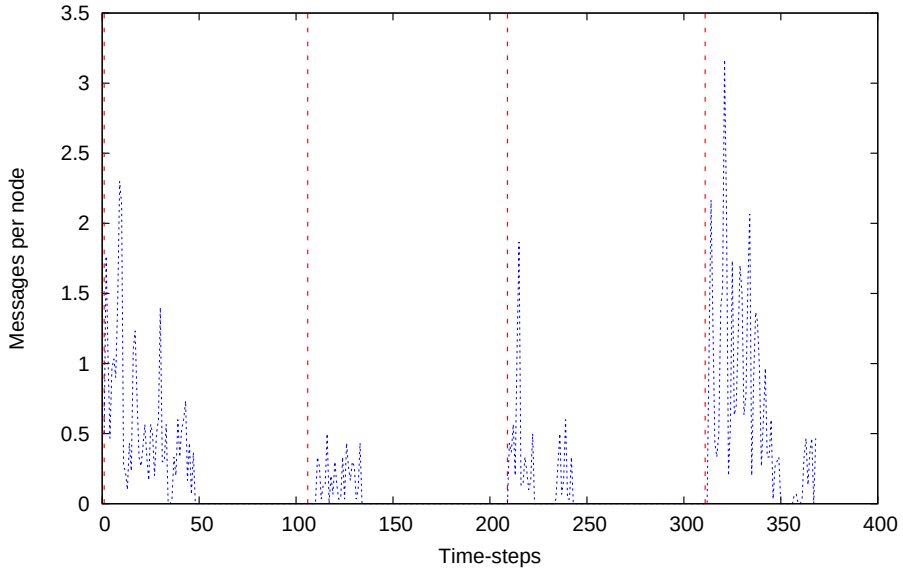


Figure 6.9: Mean number of messages used to perform the conflict resolution algorithm

are simulated. The results show that, with a small number of messages (i.e., much less than four messages per node) valid conflict-free solutions are found using the proposed approach. This means that the conflict resolution algorithm is able to

reduce conflicts in the network without negative effects on the performance of the network.

6.7 Conclusions and Future Work

In this chapter, we investigated the problem of how to include a voting mechanism to our former work to cope with the problem of conflicting trust. We highlighted the problem by using a specific example and provided an adaptive opinion mechanism in order to enable nodes in the system to individually determine their opinion values at runtime. The latter is used as basis to recognize contradicting interests among contractors when they have different trust values regarding a manager. Evaluation has been conducted to rate the effectiveness of the proposed approach and the experimental results showed that our mechanism has the advantage of decreasing conflicts among contractors with a minor increase to the amount of messages.

7

Self-Healing: Trust-Based Monitoring

Abstract. Self-healing applied to distributed systems should have the availability to continuously host services despite the failures of some of their components. As a consequence, failure monitoring plays an important role in the engineering of such systems. Many approaches in the literature make use of heartbeat messages to draw conclusions about the failure of nodes within the system. The contribution of this chapter is a heartbeat-style monitoring approach based on trust of which the benefits are twofold. On the one hand, it aims to improve the failure detection delays by adapting to the trust conditions of the network. On the other hand, it aims to reduce the number of messages that arises from sending heartbeat messages. As this technique is not limited to specific topological structures, it can be applied to many other domains. To quantify the introduced approach, evaluations have been conducted to show how such trust-based monitoring can be used to improve the efficiency of distributed self-healing systems.

7.1 Introduction

It is well-known that failure monitoring constitutes an essential building block for ensuring fault-tolerance in distributed self-healing systems [Ray05]. They pro-

vide information of nodes that have failed in the system. While the current research [MWG15, TSG+15, WYC15] focuses so far on enhancing the quality of failure monitoring, this work aims at reducing their message overhead by considering trust. The contribution of this chapter is a generic trust-based monitoring approach, which aims on the one hand to improve the failure detection delays by adapting to the trust conditions of the network and on the other hand to reduce the number of messages that arises from sending heartbeat messages. In our approach, trust can be seen as an enabler for observing the availability of monitored nodes to optimize their failure rate at runtime. The chapter offers as contribution the following aspects:

(i) an overview of current heartbeat-style monitoring approaches and their limitations in addressing the problems of large-scale distributed environments (see Section 7.2).

(ii) a formal description of the problem of how to monitor contractors effectively based on trust with the decreased usage of message overhead in distributed self-healing systems (see Section 7.3).

(iii) a decentralized trust metric covering the aspect of availability to predict the trust values of contractors, based on their uptime in the last interaction steps (see Section 7.4.1).

(iv) three round-based techniques allowing for more trustworthy contractors to be monitored less frequently than the untrustworthy ones and thus reducing the cost of message overhead at runtime (see Section 7.4.2).

All aspects are evaluated and discussed with respect to a toolkit based on the TEM [ASM+13], a trust-enabling middleware for building real-world distributed Organic Computing systems. Section 7.5 provides evaluation results of the proposed trust-based monitoring algorithm and demonstrate the benefits of the hybrid monitoring function. Finally, the chapter is closed with a conclusion and future work in Section 7.6.

7.2 Related Work

In the realm of failure monitoring a considerable volume of literature has accumulated notably in [BMS02, HDK03, HDYK04, PTT12, QZGSHK05]. One of the most popular works in this direction is the paper of Chen et al. [CTA02], which proposes a well-known technique to monitor failures based on a probabilistic analysis

of a network traffic. The authors use sampled arrival times to compute an estimation of the arrival time of the next heartbeat, where the timeout is set according to this estimation plus a constant safety margin. This approach has been extended in many directions. For example in [SPTU08], the Chen's estimation function has been replaced with another estimation based on a histogram density to compute a suspicion value of a node's failure. The histogram is then adapted whenever the system receives a new heartbeat, taking the frequencies of the last estimations into account. In [MLH+13], Miao et al. propose an online monitoring approach for silent failures, called Agnostic Diagnosis. It explores the abnormal correlation between system metrics (e.g., radio-on time, number of packets transmitted) to indicate potential silent failures at runtime. The detection accuracy of this technique is very close to 100% for small wireless sensor networks. However, it has the drawback of a central monitoring and therefore a single point of failure can occur. Our monitoring algorithm differs from state of the art heartbeat-style mechanisms in two important points: first, it can adapt to changing trust conditions of the network and second, it is not limited to specific topological structures. Furthermore, it is decentralized, generic, and applicable to any kind of trust-aware Recommender Systems. Thus, we want to compare it to systems that employ the same approach. The approach closest to us is [GMT06]. The authors propose an adaptive heartbeat approach for DHT Networks. Based on an artificial exponential distribution to model node failure, they estimate the uptime of a node and adjust its monitoring frequency accordingly. However, as exponential distributions are memoryless they cannot be used to predict uptime as we do in this work. Our algorithm can be classified as a trust-based monitoring approach. It uses the current trust condition of nodes to generate individual monitoring intervals for each contractor at runtime. Parts of the content of this chapter have been published by the author in the following conference and book chapter:

- **[MU16b]:** Nizar Msadek and Theo Ungerer. *Trust-Based Monitoring for Self-Healing of Distributed Real-Time Systems*. The 7th IEEE Workshop on Self-Organizing Real-Time Systems (SORT16) in conjunction with ISORC 2016, pages 177-178, York, England, IEEE Computer Society, 2016.
- **[MU16a]:** Nizar Msadek and Theo Ungerer. *Trust as Important Factor for Building Robust Self-x Systems*. Book chapter in Trustworthy Open Self-Organising Systems, Autonomic Systems series, Vol. 7, 2016, Springer International Publishing, pp 153-183, `http://www.springer.com/de/book/9783319291994` ISBN: 978-3-319-29199-4

7.3 Background and Contribution

In this section, we present our considered distributed system and the baseline monitoring approach that we used in our former work to detect the presence of failures.

7.3.1 System model

We focus on the class of computer systems that follow the manager-contractor paradigm of the well-known contract net protocol [Smi80]. This approach has become a standard by FIPA[1] and is often applied in many application domains for example, Organic Computing [NB09b, MKFU14, MKU15a], Self-Organizing Networks [ZZ10], or Multi-Agent Systems [NB07] etc. Such a paradigm has three main characteristics. First, it is generic and applicable to distributed environments. Second, it can work under real-time constraints [QJDL96]. And third, it can cope with the problem of scalability. In contrast to the classic master-slave paradigm in which nodes have just one role type, either master or slave [BCG04], nodes in the manager-contractor paradigm are decentralized in nature and can take on two roles, manager and contractor. Recall, we have used this form of coordination in the self-configuration Chapter 4 to assign services. This can be roughly interpreted as follows: "if a node cannot solve an assigned service due to a limited resources, it will decompose the service in sub-services and try to find other disposed nodes with the required resources to solve these sub-services". The process of assigning services is done by a contracting mechanism (see Figure 7.1) consisting of the following steps:

1. In the first step, the manager sends a contract announcement to nodes that are able to perform a certain service.
2. In the second step, the potential contractors use the description of the service and its expiration time to build bids they send to the manager.
3. Finally, the submitted bids are evaluated by the manager, which leads to awarding a service contract to the contractor with the most appropriate bid.

7.3.2 The Baseline Heartbeat Approach

Contractors play an important role in the execution of the services. Assuming that a contractor might crash, the manager should be able to detect contractor's failure and take appropriate self-healing actions, otherwise the services running on

[1] FIPA: Foundation for Intelligent Physical Agents - [Accessed: July 25, 2016] - `http://www.fipa.org/specs/fipa00029/`

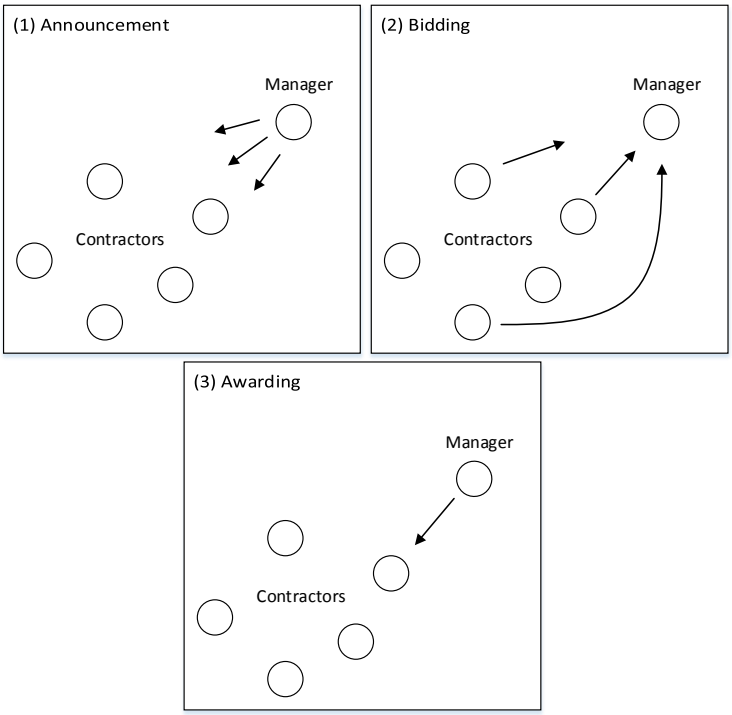


Figure 7.1: The coordination view applied by the self-configuration process to assign services. A short revision of the scheduling used in Chapter 4.

it might block the whole system. Hence, it is important for the manager to regularly monitor its contractors. A look at the literature [TVS07] reveals the two main heartbeat-style monitoring strategies: *push* and *pull* (also known in the literature as *keep-alive*, or as *hello* monitoring approaches). To see how these strategies work, assume that a manager m is monitoring a contractor c. Using a push strategy, shown in Figure 7.2, c sends at regular time interval Δ_c an alive message to m. Upon alive reception, m sets a timer $\Delta_{Timeout}$ that triggers a suspicion if it expires before the reception of a new alive message from the same contractor c. This timeout is set according to Δ_c plus a constant safety margin γ. However, in systems with a pull monitoring strategy, the monitored contractor c adopts a passive role. m monitors c by sending periodically an *Are you alive* message every Δ_c. If c does not send an answer within a certain time period of $\Delta_{Timeout}$, it gets suspected by m. Figure 7.3 illustrates how pull strategy is used for monitoring contractors. In this work the push monitoring strategy is adopted as baseline approach, since it uses only half the messages for an equivalent quality of pull monitoring approach.

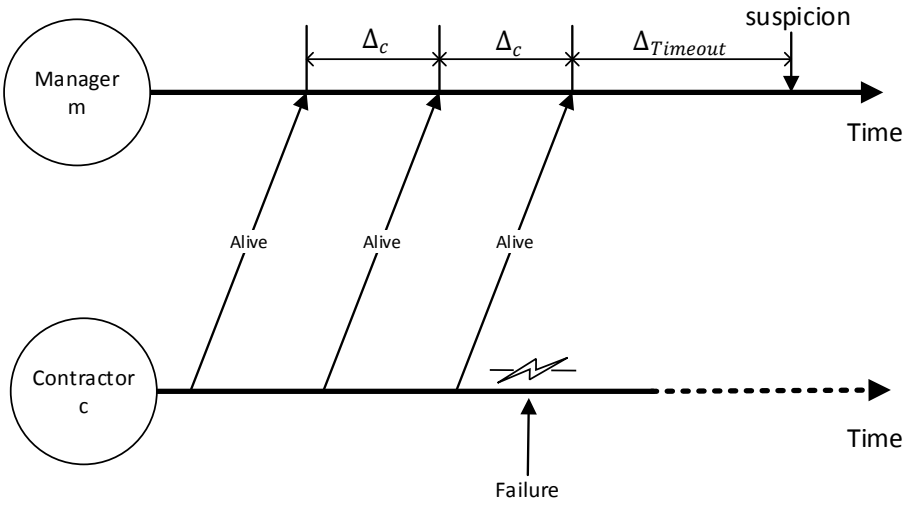


Figure 7.2: The Push monitoring strategy. Recall, this represents the current form of monitoring implemented by the self-configuration process as introduced in Chapter 4

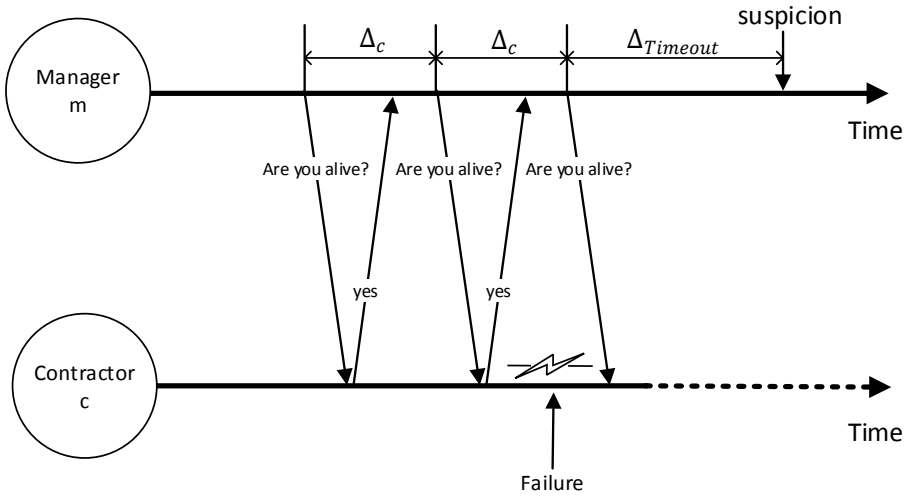


Figure 7.3: The Pull monitoring strategy

7.3.3 Manager Goal

Let $\varphi_m = \{c_1, c_2, ..., c_n\}$ be the set of contractors which are monitored by a manager m. Let $start_i^m$ denote the time at which contractor $c_i \in \varphi_m$ starts sending an alive message every Δ_c to m. Let $Uptime_i$ be the time period that c_i spends under contract. If for instance c_i leaves involuntary the network due to a failure, manager m

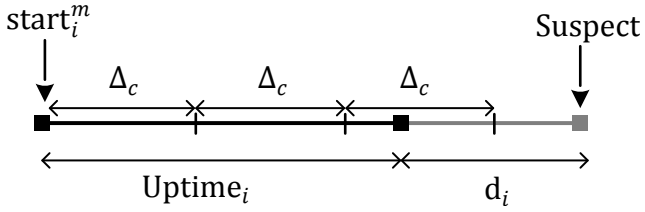


Figure 7.4: Failure detection delay of contractor c_i

would not learn of its departure until the next alive message gets suspected. As a result there is a delay d_i which m falsely believes that c_i is still under contract. This delay is called the failure detection delay (see Figure 7.4). The purpose of the manager node is twofold. On the one hand it wants to avoid or to reduce the failure detection delay of contractors during the monitoring process. On the other hand it wants to reduce the communication overhead produced by the monitoring process without the negative effect on the failure detection quality.

7.3.4 Contribution

The baseline monitoring approach makes use of parameters like $\Delta_{Timeout}$ and Δ_c to meet the current condition of the network. Adjusting these parameter values during runtime is obviously a trade-off for the manager. If for example these parameters are chosen too short, then failures are detected quickly (i.e., short delays) but more alive messages are sent in the network. A longer choice of these parameter values results in a larger failure detection delay but less communication overhead. This means that the idea to find the perfect monitoring process with the best delay and communication overhead is unrealistic. The contribution of this chapter is to help the manager to find a distributed and good enough but not necessary optimal solution to this trade-off by applying the additional notion of trust.

7.4 The Trust-Based Monitoring Approach

In this section an enhancement of the baseline monitoring approach is presented. It has been designed for systems that follow the manager-contractor paradigm as a

basis to reduce the communication overhead produced by the monitoring of contractors without the negative effect on their failure detection delays. In contrast to most existing baseline approaches[2] using a fixed uniform monitoring interval Δ_c across all contractors $<c_1, c_2, ..., c_n>$, the proposed algorithm makes use of different monitoring intervals $<\Delta_{c_1}, \Delta_{c_2}, ..., \Delta_{c_n}>$ to allow for more trustworthy contractors to be monitored less frequently than the untrustworthy ones and thus reduces the cost of message overhead. However, to detect trustworthy or untrustworthy nodes, managers need to gain knowledge about the trustworthiness of their contractors. In our approach a trust establishment process is introduced to fill the information gap of managers about the trust behavior of their contractors.

7.4.1 Trust Establishment

There are many facets of trust in computer systems [SKL+10]. Such facets may concern for example availability, reliability, credibility, safety etc. Our investigation focuses on the availability aspect. In this work it is assumed that a contractor can not realistically assess its own trust value because it trusts itself fully. Therefore, the generation of trust values must be done from the manager side. As also the cost of determining permanent trust values during runtime could negatively influence the performance of the system, we see that it is not necessary for the manager to continuously generate trust values whenever novel observations about its contractors are made. This might lead to performance bottlenecks that become more severe as the number of contractors in the system grows. For this purpose, we introduce in this work the concept of rounds. Assume again that there is a contractor c_i and let k denote the index of the round. The trust establishment process in round k is determined as follows:

$$T_{m \to c_i}^{(k)} = \begin{cases} (1-\alpha) \cdot T_{m \to c_i}^{(k-1)} + \alpha \cdot \Theta_{c_i}^{(k)} & \text{if } k \geq 1 \\ T_{init} & k = 0 \end{cases} \tag{7.1}$$

$$\Theta_{c_i}^{(k)} = \frac{\text{Uptime of } c_i \text{ in period } [k-1, k]}{\text{Total time of period } [k-1, k]} \tag{7.2}$$

$$\Theta_{c_i}^{(k)} \in [0, 1] \tag{7.3}$$

$$T_{m \to c_i}^{(k)} \in [0, 1] \tag{7.4}$$

$$k \in \mathbb{N} \tag{7.5}$$

[2]The TCP/IP keep-alive interval for Windows Server 2008 and Windows Vista is set by default to 120 minutes — [Accessed: July 25, 2016] — `https://technet.microsoft.com/de-de/library/dd349797.aspx`

Manager m calculates a trust value $T_{m \to c_i}^{(k)}$ for c_i based on the new observation $\Theta_{c_i}^{(k)}$ and the previous trust value $T_{m \to c_i}^{(k-1)}$ (see Equation 7.1). This trust value $T_{m \to c_i}^{(k)}$ is always within $[0, 1]$ and reflects the subjective trust of manager m in contractor c_i. A trust value of $T_{m \to c_i}^{(k)} = 0$ means m does not trust c_i at all while a value of 1 stands for whole trust. The factor $\alpha \in [0, 1]$ decides how strong the recent observations are weighted compared to the previous ones. The larger the value α, the more the result is computed by the recent observations. Initially, the trust value of each contractor is set to T_{init} and in every round k an update occurs for $T_{m \to c_i}^{(k)}$.

7.4.2 Individual Monitoring Intervals

The calculation and adjustment of a contractor monitoring interval $\Delta_{c_i}^{(k)}$ in every round k is certainly a crucial part for the manager. By extending each manager with a trust component and modeling the relations between a manager and its contractors with a trust mechanism, we are able to calculate these intervals.

Discrete Monitoring

One simple way to make this calculation is to classify the monitoring intervals by using discrete frequencies. As shown in Figure 7.5, a constant value T_{freq} is defined for frequently monitoring and another constant value T_{infreq} for infrequently monitoring, i.e., with $T_{freq} < T_{infreq}$. If for example a manager m is monitoring a trustworthy contractor c_i (i.e., with a trust value $T_{m \to c_i}^{(p)}$ above a threshold δ) the monitoring interval $\Delta_{c_i}^{(k)}$ in round k can be set to T_{infreq}, otherwise it is set T_{freq}. In detail, the discrete monitoring function is calculated using Equation 7.6.

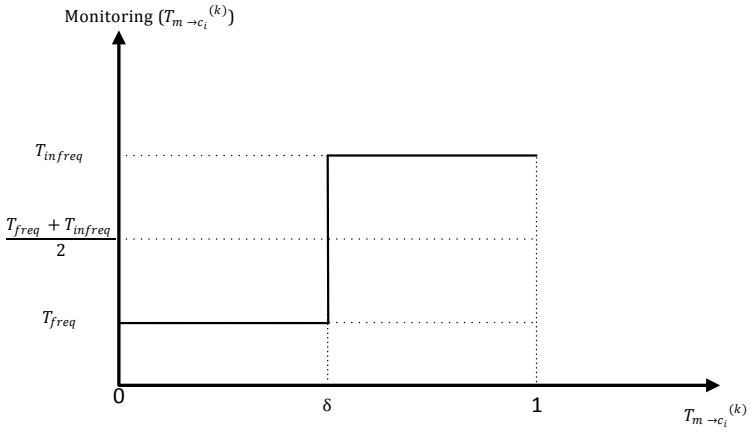


Figure 7.5: Discrete monitoring function

$$Monitoring(T^{(k)}_{m\to c_i}) = \begin{cases} T_{freq} & \text{if } 0 \leq T^{(k)}_{m\to c_i} \leq \delta \\ T_{infreq} & \text{if } \delta < T^{(k)}_{m\to c_i} \leq 1 \end{cases} \tag{7.6}$$

Continuous Monitoring

However, it might be critical to distinguish only between T_{freq} and T_{infreq} monitoring intervals, especially for contractors whose trust values are very close to the defined threshold δ. We therefore examine another classification function with continuous monitoring values.

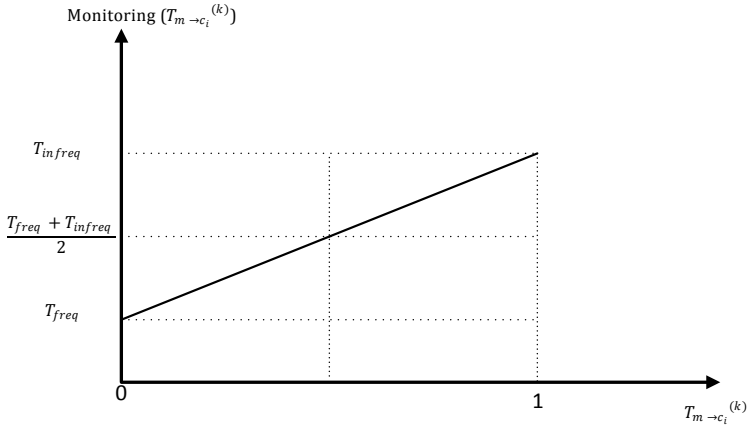


Figure 7.6: Continuous monitoring function

Figure 7.6 depicts an overview of this function. It is easy to see that the monitoring function is always continuous on the interval $[T_{freq}, T_{infreq}]$. The larger the value of $T^{(k)}_{m\to c_i}$, the less frequently the resulted monitoring. In detail, the continuous monitoring function is calculated using Equation 7.7.

$$Monitoring(T^{(k)}_{m\to c_i}) = (T_{infreq} - T_{freq}) \cdot T^{(k)}_{m\to c_i} + T_{freq} \tag{7.7}$$

Continuous-Discrete Monitoring

The monitoring function can be further improved if the trust specification of the system is taken into account. Therefore, a metric is examined which consists of a combined continuous-discrete monitoring function with the ability to adapt the trust threshold δ of the manager. The basic idea of this metric is quite simple. If the trust value $T^{(k)}_{m\to c_i}$ is lower than the threshold δ, then $\Delta^{(k)}_{c_i}$ is determined continuously from the interval $[T_{freq}, \frac{T_{freq}+T_{infreq}}{2}]$. It is always assumed that $T_{freq} < T_{infreq}$.

If, however, the value $T^{(k)}_{m\to c_i}$ crosses δ, then c_i is monitored less frequently with a continuous value from the interval $[\frac{T_{freq}+T_{infreq}}{2}, T_{infreq}]$. In detail, the continuous-discrete monitoring function is calculated using Equation 7.8.

$$Monitoring(T^{(k)}_{m\to c_i}) = \begin{cases} (\frac{T_{infreq}-T_{freq}}{2\delta}) \cdot T^{(k)}_{m\to c_i} + T_{freq} & \text{if } 0 \leq T^{(k)}_{m\to c_i} \leq \delta \\ (\frac{T_{infreq}-T_{freq}}{2(1-\delta)}) \cdot T^{(k)}_{m\to c_i} + T_{infreq} - \frac{T_{infreq}-T_{freq}}{2(1-\delta)} & \text{if } \delta < T^{(k)}_{m\to c_i} \leq 1 \end{cases} \tag{7.8}$$

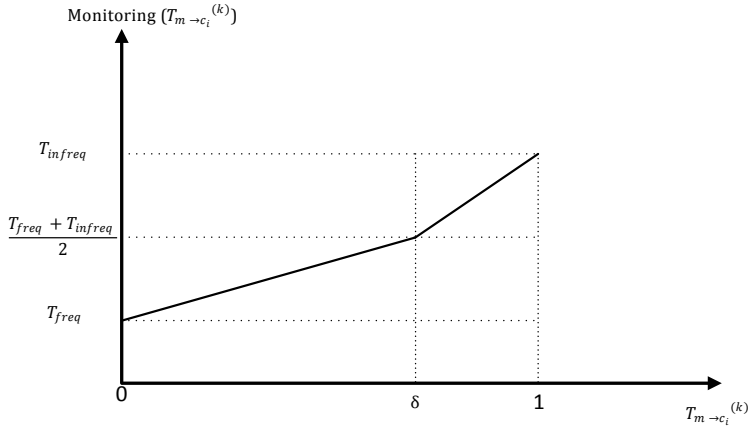


Figure 7.7: A combined continuous-discrete monitoring function

7.5 Evaluation

7.5.1 Evaluation Setting

In this section a methodology is introduced to evaluate the proposed monitoring approaches. For this purpose, an evaluator based on TEM [ASM+13] for heartbeat-style monitoring approaches has been implemented. Using this tool, a series of measurements have been conducted to explore the limit performance of the presented monitoring approaches and to compare them to the theoretical optimal result. All the evaluated approaches make use of the same network setting to create fairness judgements. This consists of a network with 100 nodes, where all nodes are able to communicate with each other using message passing. The number of operating managers is set to 10 while every manager is responsible for monitoring 10 contractors. Each contractor has a limited resource capacity and an individual trust value without any central knowledge, as shown in Table 7.1. The initial service assignment is performed randomly to ensure that the proposed approaches

Type	CPU (MHz)	Memory (MB)	Trustworthiness	Amount%
Type 1	200-800	500-1000	0.7-0.9	10
Type 2	500-1500	500-1500	0.3-0.6	50
Type 3	1500-2000	2000-4000	0.4-0.8	30
Type 4	2000-3000	4000-8000	0.4-0.9	10

Table 7.1: Heterogeneous contractors

are evaluated under a great variety of start conditions. After the assignment, the manager starts to monitor its contractors using the following monitoring setting:

- The frequently monitoring parameter T_{freq} is set to 50 (in Time Units)
- The infrequently monitoring parameter T_{infreq} is set to 2000 (in Time Units)
- The constant safety margin γ is set to 10% of Δ_c

In the conducted evaluation, it is assumed that the crash of contractors would never occur at once but rather separately with a MTBF= 10000 in Time Units. Therefore, a selection metric is used to decide which contractor fails at each time step. This selection metric is based on a roulette wheel selection, where contractors with lower trust values (untrustworthy contractors) have a higher chance to fail than other contractors with a higher trust values (trustworthy contractors). The basic methodology for investigating the monitoring approaches contains the following two steps:

Exploring evaluation: In the first step an exploring investigation process is used to examine the performance limit of each individual monitoring approach using different tuning parameters.

Comparative evaluation: Based on the results of the exploring evaluation, a comparative performance analysis for the monitoring approaches is investigated and thus for best solutions they are providing.

7.5.2 Exploring evaluation

The first barrier to compare the monitoring approaches are their different tuning parameters. These influence the time when a manager starts to suspect a failure and its subsequent message cost. To be able to compare later the monitoring approaches fairly, the behavior of each one of them has to be explored using different parametrization. The first two Figures 7.8 and 7.9 examine the baseline monitoring using different monitoring frequencies. As would be expected, by reducing the

size of the monitoring frequency to T_{freq} (frequent monitoring), the average failure detection delay is decreased but more overhead costs are incurred in the network. However, by extending the monitoring frequency to T_{infreq} (infrequent monitoring) the message overhead is reduced but a larger failure detection delay is resulted. Therefore, there is a trade-off between the failure detection delay and the message cost. A theoretical optimum is obtained by taking the best value of each criteria (i.e., the lowest value for detection delay $O^*_{delay} = 78$ and lowest value for message overhead $O^*_{cost} = 299$). It is obvious that this optimum could never be reached in the practice. However, we use this optimum as a lower bound in Section 7.5.3 to compare the monitoring approaches. For the discrete monitoring the tuning parameter is in this case the threshold δ. Therefore, experiments are conducted in Figures 7.10 and 7.11 by varying the amount of δ. The results showed that the larger the choice of δ, the earlier the detection of failure will be. However, if δ is chosen too large a huge amount of messages occurs. The continuous monitoring approach has no tuning parameter. It is only one single point in the evaluation results, i.e., it took on average 1416 for failure detection delay and 682 for message overhead. Finally, the behavior of the continuous-discrete monitoring is explored with different amount of δ. The results in Figures 7.12 and 7.13 showed that the hybrid monitoring attests very good detection delays compared to the other approaches.

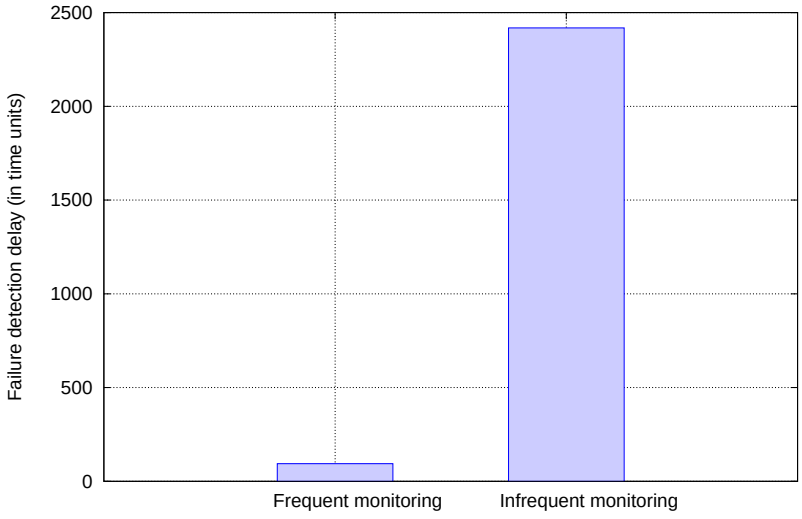


Figure 7.8: Failure detection delay for the baseline monitoring approach using different monitoring frequencies.

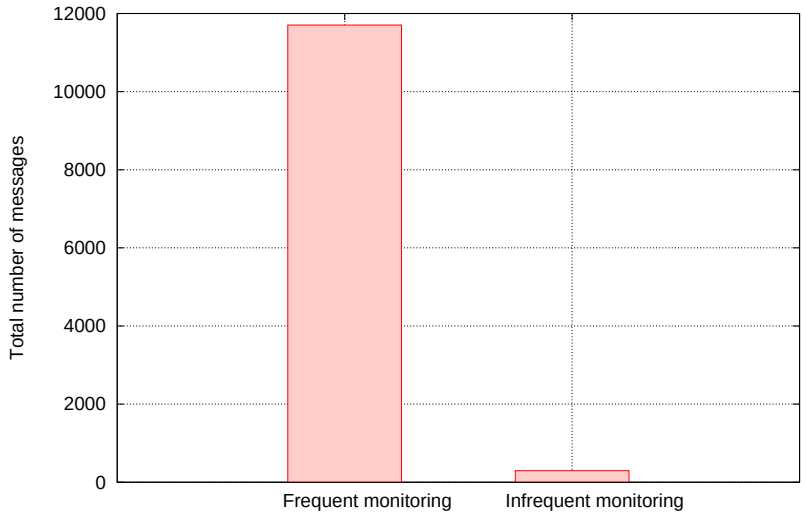


Figure 7.9: Message overhead for the baseline monitoring approach using different monitoring frequencies.

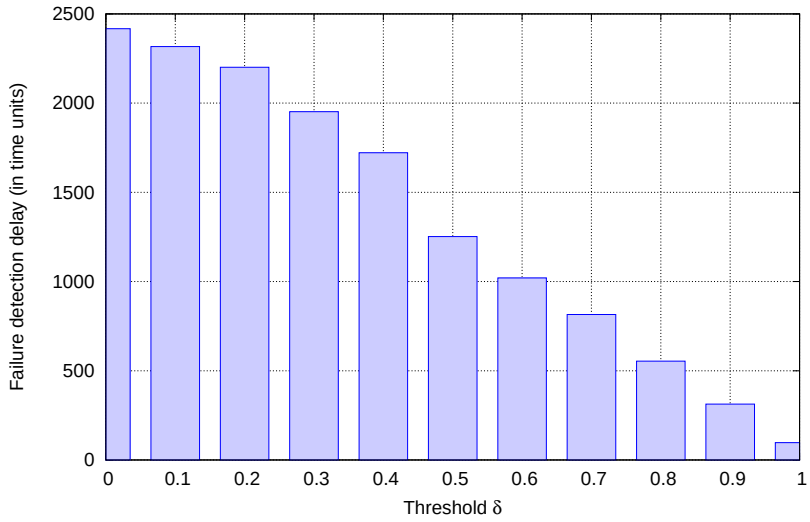


Figure 7.10: Failure detection delay for the discrete monitoring approach using different amount of δ.

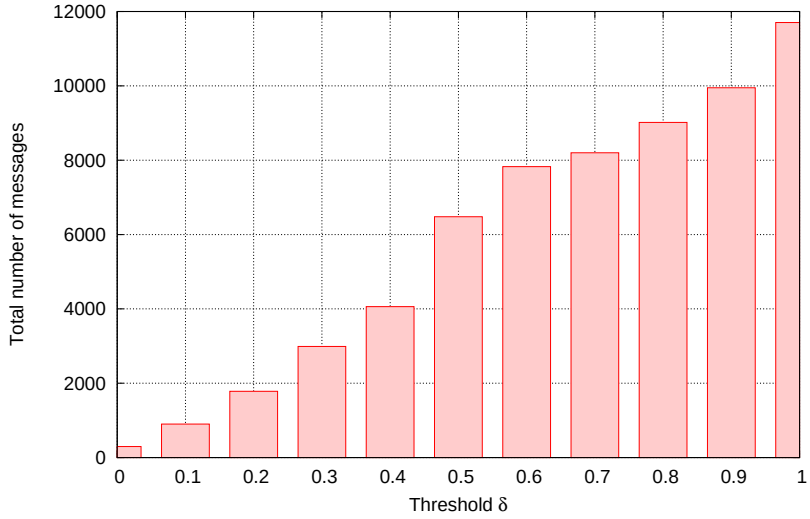


Figure 7.11: Message overhead for the discrete monitoring approach using different amount of δ.

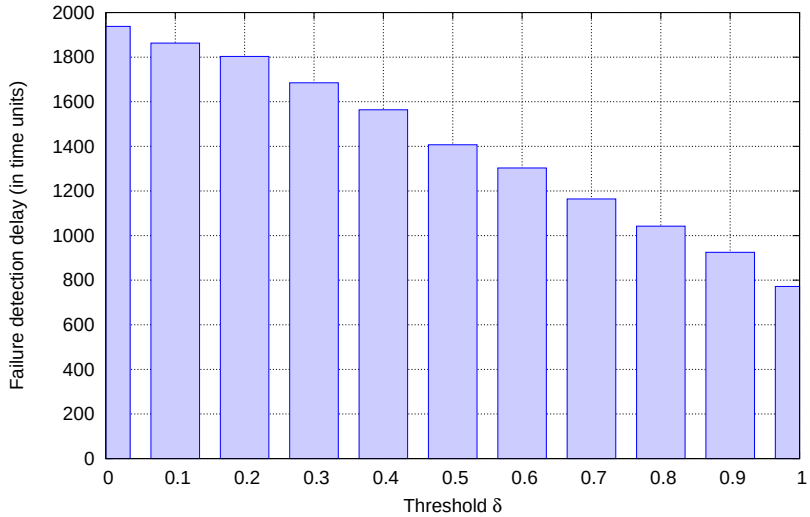


Figure 7.12: Failure detection delay for the continuous-discrete monitoring approach using different amount of δ.

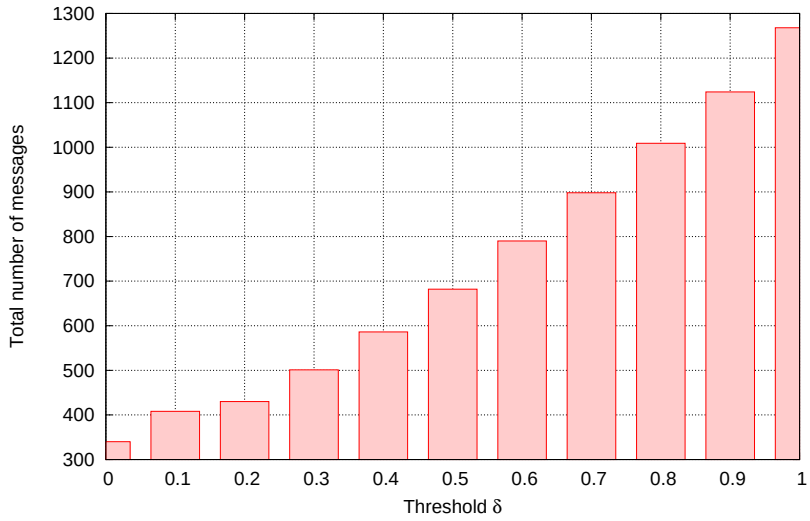


Figure 7.13: Message overhead for the continuous-discrete monitoring approach using different amount of δ.

7.5.3 Comparative evaluation

In this section the monitoring approaches are compared regarding the best solutions they are providing. The result of this experiment is depicted in Figure 7.14, which shows the average failure delay on the horizontal axis and the message overhead on the vertical axis. Values near to the theoretical optimum represent a short detection delay with few messages. To quantify the reached performance of each algorithm, the Euclidean Distance is used. Formula 7.9 describes how the latter is calculated. O^*_{delay} and O^*_{cost} are lower bounds for delay and cost of the theoretical optimum estimated a priori in Section 7.5.2. S_{delay} and S_{cost} represent the reached delay and cost for the current used monitoring approach. Applying this formula to our experiment leads to the following results:

- $d_{frequent} = 11410$
- $d_{infrequent} = 2339$
- $d_{discrete} = 2589.67$
- $d_{continuous} = 1391.73$
- $d_{continuous-discrete} = 1182.38$

Concluding, the result shows that the continuous-discrete monitoring attest the shortest distance from the theoretical optimum. It performs better than the other algorithms and therefore considered suitable for our test scenario.

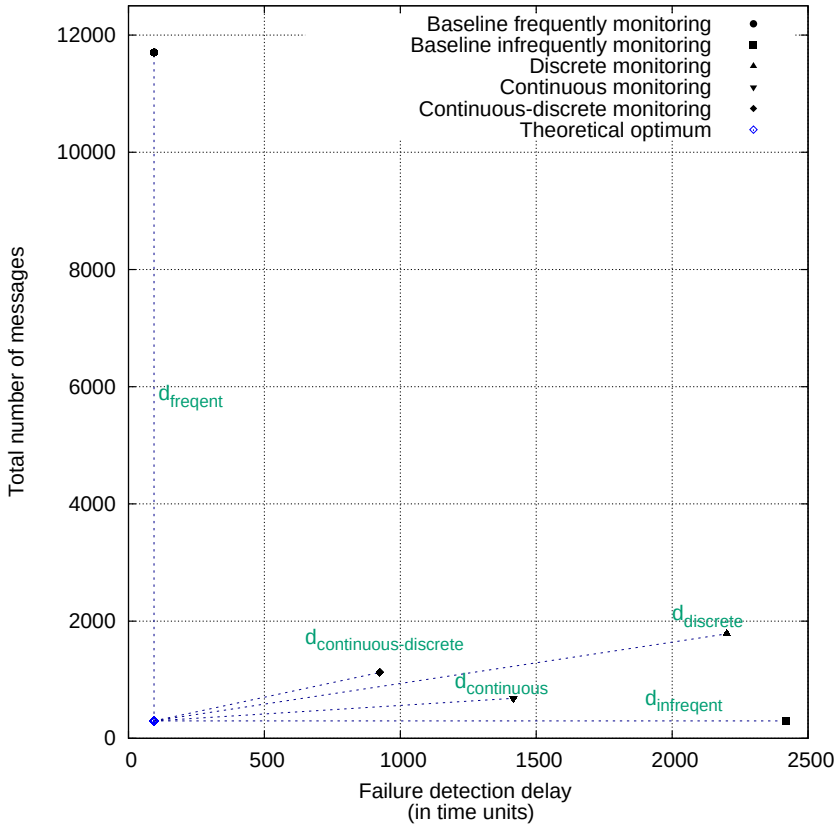


Figure 7.14: Best found solutions compared to the theoretical optimum

$$dist((O^*_{delay}, O^*_{cost}), (S_{delay}, S_{cost})) = \sqrt{(O^*_{delay} - S_{delay})^2 + (O^*_{cost} - S_{cost})^2} \tag{7.9}$$

7.6 Conclusions and Future Work

In this chapter, we examined the problem of how to monitor contractors effectively based on trust with the decreased usage of message overhead in distributed self-healing systems. We formalized the problem, expressed it analytically in form of a tractable trade-off problem in Section 7.3, and derived a trust-based monitoring solution for it. More precisely, a trust metric has been developed covering the aspect of availability to predict the trust values of contractors at runtime, based on their uptime in the last interaction steps (see Section 7.4.1). This is used to generate different monitoring intervals allowing more trustworthy contractors to be monitored less frequently than the untrustworthy ones and thus reducing the cost of message overhead. To do so, three monitoring functions have been introduced in Section 7.4.2: discrete monitoring, continuous monitoring, and continuous-discrete monitoring. The results of Section 7.5 show that the continuous-discrete monitoring performs best. It can adapt faster to changing trust condition in the network than the others. Interesting starting point for future work is to bound the failure detection delay to allow for rescheduling the services within a certain time period set by the deadline.

8

Self-Healing: Trust-Based Replication

Abstract. Replication occurs in a wide range of open systems, ranging from Organic and Autonomic Computing to many other Multi-Agent Computing Systems. All these systems face a common issue: how can replicas automatically and efficiently be managed in a system despite changing requirements of their environment? One way to overcome this issue is *trust*. The contribution of this chapter is a novel approach based on trust that provides a good management of replicas — especially for those of important services — despite malicious behavior of nodes in the network. Depending on the importance level of a service possessing the replicas and the assessment of the trustworthiness of a node, we can optimize the trust distribution of replicas at runtime. For evaluation purposes we applied our approach to an evaluator based on the TEM middleware. In this testbed, the usage of trust reduced the replication overhead by about 14% while providing a much better placement of important replicas than without trust.

8.1 Introduction

Implementing replication mechanisms for practical systems like Organic and Autonomic Computing is a non-trivial task. This is due to the fact that such systems are becoming increasingly complex in their organisational structures, especially when unknown heterogeneous entities might arbitrarily enter and leave the network at

any time. Therefore, new ways have to be found to develop and manage them. One way to overcome this issue is trust. Using appropriate trust mechanisms, entities in the system can have a clue about which entities to cooperate with. This is very important to enhance the robustness of such systems, which depend on a cooperation of autonomous entities. In this chapter, we primarily focus on self-healing and note that our goal is to develop an autonomous replication mechanism based on trust that works in a distributed manner and also ensures global optimality. The mechanism should provide a good replica management and placement — especially for those of important services — despite malicious behavior of nodes in the network. Therefore, the contribution of this chapter leads to a methodology that offers:

(i) An overview of the system model and the considered problem to indicate the specific purpose of our research (see Sections 8.2 and 8.3),

(ii) a mechanism for specifying the importance level of services based on the number of requests as well as a mechanism to monitor the trust behavior of nodes at runtime (see Section 8.4.1), and

(iii) a replication model considering the above introduced points to manage the amount of replicas for better trustworthiness of important services (see Section 8.4.2)

All aspects are evaluated and discussed with respect to a toolkit based on the TEM [ASM+13], a trust-enabling middleware for building real-world distributed Organic Computing systems. Section 8.5 provides evaluation results of the proposed mechanism and Section 8.6 aligns the trust-based approach in the context of state of the art systems. Finally the chapter is closed with a conclusion and future work in Section 8.7.

8.2 System Model and Assumptions

We target a distributed system consisting of a finite set of nodes $\mathcal{N} = \{n_1, n_2.., n_n\}$, representing machines which can interact with each other through a set of messages. The i-th node is denoted by n_i, or alternatively by i if it is not ambiguous. These nodes are heterogeneous in terms of storage resources. Thus, every node provides a storage space with a free capacity $\texttt{Capa}_i$ to offer services in the system a place for storing their data. The set of services is denoted by $\mathcal{S} = \{s_1, s_2.., s_k\}$ and the data of a service s_j is expressed by $\texttt{Data}_j$. This data is replicated among the nodes of the network. Thus, a $\texttt{Data}_j$ is partitioned into a set of owner and replicas with $\texttt{Data}_j = \{\texttt{Owner}_j\} \cup \{\texttt{Repl}_{j,1}, \texttt{Repl}_{j,2}, ..., \texttt{Repl}_{j,r_j}\}$. The $\texttt{Owner}_j$ is the central

element that performs all write requests. It has multiple replicas which replicate its data. We assume that these replicas are completely identical to the owner in contrast to erasure codes [1]. This means that the storage consumptions for all replicas are the same, which is given by the value $\texttt{Consum}_j$. Whenever an $\texttt{Owner}_j$ performs a write request, it delivers an update with a new version of its data to the replicas. These replicas are used to perform read requests and to overcome node failures. Their amount is fixed so far as a system parameter r_j within the interval $[R_{min}, R_{max}]$ to avoid that not too many but also not too few replicas are created. Should an owner of a data service fail, the system will elect one of the replicas with the latest version to takeover the role of owner. We also assume that the number of requests may affect the importance level of services. Services having a large amount of requests are considered to be important for the functionality of the entire system. From this point of view, important services are supposed to be rational in the sense that they want to place their replicas as well as possible in the system, by choosing only high trustworthy nodes.

8.3 Problem Statement and Baseline

A crucial point in our system is the trustworthiness of the service storage, especially for important services. Their data should be hosted on trustworthy nodes having a high degree of trustworthiness despite malicious behavior of nodes in the network. Therefore, we formulate the trustworthy replication management problem as follows. Given a set of services with different importance levels and a network topology with a finite number of nodes representing possible replica locations, we are interested to determine a trustworthy placement for replicas such that the trustworthiness of important data is improved by hosting them on trustworthy nodes. On the other hand, the storage of nodes should be optimized in terms of resource consumptions as well. To do so, four fundamental decisions have to be considered: (1) which services are considered as more important than others in the system, (2) when should the categorization of services be determined, (3) how to fix the right amount of replicas, and (4) where to place these replicas on nodes. Depending on these decisions, different replication strategies can be applied. Figure 8.1 shows the baseline replication strategy considered in this work. It is based on thresholds for determining the importance level of services. These thresholds are fixed at design time and do not change during run time. Then, a scheduler is used for creating and

[1] An erasure code provides redundancy without saving the identical complete copy of an object element. It divides the element into m fragments and recodes them into n fragments, where $n > m$. Details can be found in the paper referenced here [WK02].

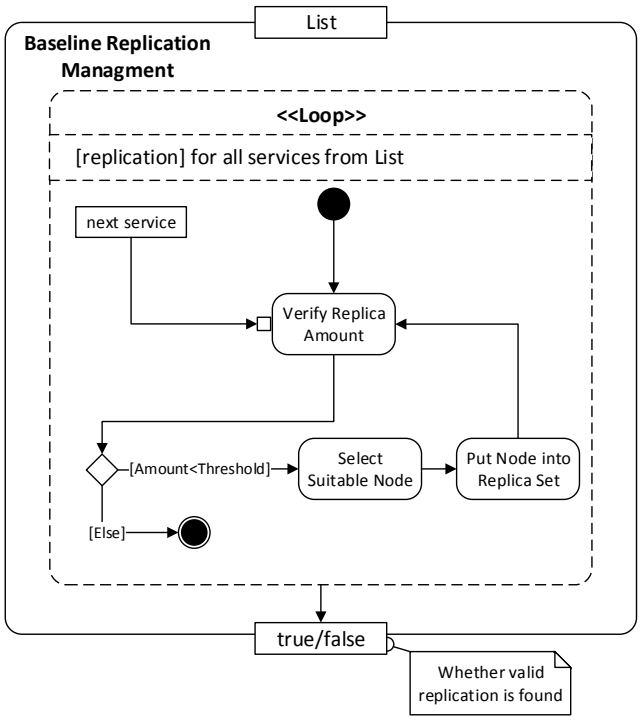


Figure 8.1: Elementary representation of the baseline replication management

assigning replicas. For every service, it selects a node with a uniform probability without considering its trust value or up probability and put it into a replica set. If for example a node is already selected or its storage space is full, it picks another new node, until the given number of replicas reaches the threshold R_{max} for important services or the threshold R_{min} for unimportant services. This strategy has the ability to improve the availability of service data in the system. However, it is only suited for classical systems where the benevolence of nodes is assumed [SFSR94], because all nodes are treated identically even though some of them are less trustworthy than others. In open distributed systems, it would not provide good results due to the fact that unknown nodes might arbitrarily enter and leave the network at any time. Therefore, we propose to incorporate trust mechanisms into the management of replicas.

8.4 The Trust-Based Replication Approach

8.4.1 Models and Metrics

Our aim is to provide a continuous good trustworthiness for data especially for important services and to reduce the performance overhead produced by creating

too many replicas in the system as well. As a consequence, our approach should consider the following factors: (1) We want a mechanism for specifying the required trust of services based on their number of requests. (2) We need a mechanism to regularly monitor the trust behavior of nodes. (3) And we want a model to manage the amount of replicas for better trustworthiness of important services. It is worth mentioning that all solutions provided should always be round-based over the time to ensure that the system is continuously optimized at runtime.

Determine the Required Trust of Services

As mentioned, the number of requests plays a crucial role in the categorization of services. Services with a large number of requests are considered to be more important than others in the system. They should have a higher degree of required trust in order to be hosted on trustworthy nodes. The baseline approach does not take such a decision into account at runtime. It categorizes the importance level of services at the beginning which is then not changed during execution. Our aim is to give the system more responsibility by moving this decision from design-time to runtime. The algorithm for this strategy consists of two phases which are bounds calculation and required trust calculation phase. In the first phase, we compute the cumulative number of requests for each service $s_j \in \mathcal{S}$ at every round k as follows:

$$Request^{k}_{(s_j)} = Request^{k}_{(\texttt{Owner}_j)} + \sum_{i=1}^{r_j} Request^{k}_{(\texttt{Repl}_{j,i})} \tag{8.1}$$

Where $Request^{k}_{(\texttt{Owner}_j)}$ stands for the number of read and write requests performed by the owner until round k is reached, and $Request^{k}_{(\texttt{Repl}_{j,i})}$ represents the total number of read requests that were performed by every replica i of s_j until k is reached. Then, we determine the minimum and maximum values over these requests as follows:

$$\texttt{MinRequest}^{\texttt{k}} = \{Request^{k}_{(s_j)} | \forall s_i, s_j \in \mathcal{S} : Request^{k}_{(s_i)} \geq Request^{k}_{(s_j)}\} \tag{8.2}$$

$$\texttt{MaxRequest}^{\texttt{k}} = \{Request^{k}_{(s_j)} | \forall s_i, s_j \in \mathcal{S} : Request^{k}_{(s_i)} \leq Request^{k}_{(s_j)}\} \tag{8.3}$$

By this means, the request number of every service is always bounded between $\texttt{MinRequest}^{\texttt{k}}$ and $\texttt{MaxRequest}^{\texttt{k}}$. Values near to the $\texttt{MinRequest}^{\texttt{k}}$ reflect unimportant services, whereas values close to $\texttt{MaxRequest}^{\texttt{k}}$ stand for important services. In the second phase, we aim to compute the required trust of each service. Assume that $\texttt{MinTrust}$ and $\texttt{MaxTrust}$ are the desired thresholds for minimum and maximum trust set in the system, where $0 \leq \texttt{MinTrust} \leq \texttt{MaxTrust} \leq 1$. Then, a value of $Request^{k}_{(s_j)}$ can be mapped to a required trust value $ReqTrust^{k}_{(s_j)}$ in the new specified

range $[\texttt{MinTrust}, \texttt{MaxTrust}]$ using min-max normalization as follows:

$$ReqTrust^{k}_{(s_j)} = \frac{Request^{k}_{(s_j)} - \texttt{MinRequest}^{\texttt{k}}}{\texttt{MaxRequest}^{\texttt{k}} - \texttt{MinRequest}^{\texttt{k}}} \cdot (\texttt{MaxTrust} - \texttt{MinTrust}) + \texttt{MinTrust} \tag{8.4}$$

We give a high degree of trust for important services by shifting the minimum and maximum number of requests to MinTrust and MaxTrust, respectively. Figure 8.2 shows – as example – the required trust scores that can take services s_i and s_j after min-max normalization. The original distribution of requests is retained for both services and is then transformed in required trust values in the new specified range of $[\texttt{MinTrust}, \texttt{MaxTrust}]$.

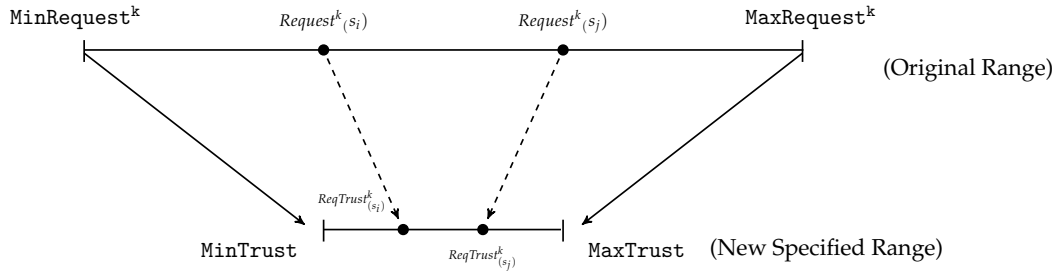


Figure 8.2: Mapping of requests from the range of $[\texttt{MinRequest}^{\texttt{k}}, \texttt{MinRequest}^{\texttt{k}}]$ to required trust in a new specified range of $[\texttt{MinTrust}, \texttt{MaxTrust}]$ using min-max normalization.

Assess the Trust Behavior of Nodes

A trust-based replication needs a component which generates trust values based on direct experiences to detect the presence of untrustworthy nodes in the system. According to [SKL+10], this trust generation can be measured by regarding different facets of trust, such as reliability, availability, functional correctness, and usability. The facet of interest in this work is availability, since we are interested to check the availability of nodes based on their uptime in the last interaction steps. Also we assume that the trust relation *Trust* between nodes is always irreflexive on $\mathcal{N}$, i.e., $\forall n_i \in \mathcal{N} :\Leftrightarrow \neg(n_i \; Trust \; n_i)$ meaning that we do not allow the possibility for nodes to assess their own trust values. Otherwise, nodes would trust themselves fully and the system will be prone to exploitation from malevolent nodes. Equation 8.5 shows the metric we set to calculate direct trust using the facet availability. It corresponds mainly to the same metric of Section 7.4.1, but the focus here is set on replication purposes.

$$Trust^{k}_{(n_i,n_j)} = \begin{cases} (1-\alpha) \cdot Trust^{k-1}_{(n_i,n_j)} + \alpha \cdot \Theta^{k}_{n_j} & \text{if } k \geq 1 \\ Trust_{init} & k = 0 \end{cases} \tag{8.5}$$

$$\Theta^{k}_{n_j} = \frac{\text{Uptime of } n_j \text{ in period } [k-1,k]}{\text{Total time of period } [k-1,k]} \tag{8.6}$$

$$\Theta^{k}_{n_j} \in [0,1] \tag{8.7}$$

$$Trust^{k}_{(n_i,n_j)} \in [0,1] \tag{8.8}$$

$$k \in \mathbb{N} \tag{8.9}$$

In every round k, a node n_i calculates a trust value $Trust^{(k)}_{(n_i,n_j)}$ about its direct neighbor n_j based on the new observation of $\Theta^{(k)}_{n_j}$ and the previous trust value $Trust^{(k-1)}_{(n_i,n_j)}$. This trust value $Trust^{(k)}_{(n_i,n_j)}$ ranges always within $[0,1]$ and reflects the subjective trust of node n_i in node n_j based on its experiences. A trust value of $Trust^{(k)}_{(n_i,c_i)} = 0$ means n_i does not trust n_j at all while a value of 1 stands for full trust. The factor $\alpha \in [0,1]$ decides how strong the recent observations are weighted compared to the previous ones. The larger the value α, the more the result is computed by the recent observations. Initially, the trust value of n_j is set to $Trust_{init}$ and in every round k an update occurs for $Trust^{(k)}_{(n_i,n_j)}$. With increasing number of mutual interactions over k, we expect to correctly estimate the behavior of nodes in the system. This is very important to prevent the hazardous placements of replicas on nodes.

Perform the Placement of Replicas

A crucial point in the baseline algorithm is the trustworthy placement of service data. To guarantee a specific level of trustworthiness the nodes hosting the replicas have to be chosen correctly. An optimal selection of these nodes is not considered in the baseline version. Therefore, we are interested in our approach to improve the replica placement by considering the trust behavior of each replica. Let us assume the required trust of a service s_j is known and that $\texttt{Owner}_j$ has multiple replicas $\{\texttt{Repl}_{j,1}, \texttt{Repl}_{j,2}, ..., \texttt{Repl}_{j,r_j}\}$ which replicate its $\texttt{Data}_j$. These replicas are distributed on different nodes and the trust values of nodes are independent of each other. The probability of $\texttt{Data}_j$ to be in a trustworthy state at round k is represented by $Trust^{k}_{(\texttt{Data}_j)}$ and its untrustworthiness is denoted by $\overline{Trust^{k}_{(\texttt{Data}_j)}} = 1 - Trust^{k}_{(\texttt{Data}_j)}$. It is obvious that $\texttt{Data}_j$ is in an untrustworthy state if and only if all replicas as

well as the owner are not trustworthy. So the untrustworthiness of $\texttt{Data}_j$ can be determined by Equation 8.10.

$$\overline{Trust^k_{(\texttt{Data}_j)}} = \overline{Trust^k_{(\texttt{Owner}_j)}} \cdot \prod_{i=1}^{r_j} \left(\overline{Trust^k_{(\texttt{Repl}_{j,i})}} \right) \tag{8.10}$$

This means that the probability of at least one replica to be in a trustworthy state can be written as follows:

$$Trust^k_{(\texttt{Data}_j)} = 1 - \left(\overline{Trust^k_{(\texttt{Owner}_j)}} \cdot \prod_{i=1}^{r_j} \left(\overline{Trust^k_{(\texttt{Repl}_{j,i})}} \right) \right) \tag{8.11}$$

To ensure that this probability is always greater than or equal to $ReqTrust^k_{(s_j)}$, we make use of Equation 8.12 as a condition.

$$1 - \left(\overline{Trust^k_{(\texttt{Owner}_j)}} \cdot \prod_{i=1}^{r_j} \left(\overline{Trust^k_{(\texttt{Repl}_{j,i})}} \right) \right) \geq ReqTrust^k_{(s_j)} \tag{8.12}$$

By this means, solving the placement problem consists of minimizing the amount of replicas r_j needed such that the constraint of Equation 8.12 is met. This is very important to prevent the hazardous placements of replicas on nodes and to reduce the replication overhead during runtime.

8.4.2 The Approach

The interaction between the activities of the trust-based replication approach is illustrated in Figure 8.3. In the first step, we make use of the trust metric of Equation 8.5 — by regarding availability as a facet of trust — to assess the behavior of nodes. Our metric takes the advantage to converge to the true hidden trust values of nodes with increasing number of mutual interactions over k. This is very important to detect trust anomalies in node behavior and to allow owners to decide where to place their replicas trustworthily in the system. Then an update is initiated at every round k to refresh the trust values of nodes as well as the recognition of important services at runtime. Services with a large amount of requests are considered to be important for the functionality of the entire system. As a consequence, we give them a high degree of required trust using Equation 8.4. Then, replicas are managed for every service so that the condition of Equation 8.12 will hold. This replication management is accomplished by removing, replacing or adding the minimum number of replicas for every service. The smaller the amount of replicas is, the lower the replication cost and the better the performance of the system will be. Using trust, our replication management has the following benefits compared to conventional replication systems: On the one hand it reduces the overall

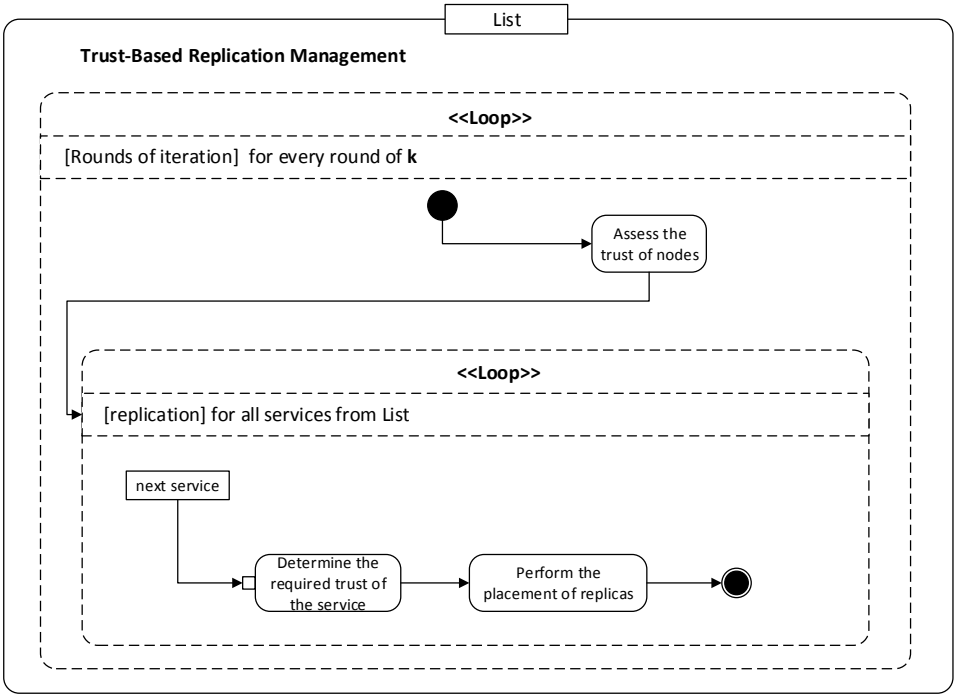


Figure 8.3: Elementary representation of the trust-based replication management

number of replicas produced in the system. This optimization cost is continuously performed over the system lifetime. On the other hand, it improves the trustworthy placement of replicas on nodes so that the more important replicas will be always placed only on highly trustworthy nodes.

8.5 Evaluation

In this section, we investigate the effectiveness of the trust-based replication approach. For the purpose of evaluating and testing, an evaluator based on our TEM [ASM+13] middleware has been implemented which is able to simulate our approach over a period of 2000 rounds. The evaluation network consists of 1000 nodes where all nodes are able to communicate with each other using message passing. Each node has a random storage capacity and is judged by an individual trust value without any central knowledge. The trust values of nodes are generated in two steps. Firstly, according to [BLF09, RDB10], we created different behaviors of nodes with different proportions in the system:

- **Durable trustworthy:** with a mean value of 0.95 and proportion of 10%
- **Stable trustworthy:** with a mean value of 0.87 and proportion of 25%

- **Unstable trustworthy:** with a mean value of 0.75 and proportion of 30%
- **Erratic** : with a mean value of 0.33 and proportion of 35%

Then, we added a Gaussian noise of $\sigma = 0.1$ to each trust and capped the resulting value into $[0, 1]$. The evaluation has been conducted using 100 services (i.e., 50% of them are important and 50% unimportant). The service assignment has been performed randomly to ensure that the proposed approach is evaluated under a great variety of start conditions. After the assignment, replicas are created in the system using the following parameters:

- The minimum replication factor R_{min} is set to 5.
- The maximum replication factor R_{max} is set to 20.

Our goal is to recognize the importance level of services solely based on the number of requests and to improve the trustworthy placement of replicas at runtime. Furthermore, replication cost should be reduced in contrast to the baseline approach in order to get a better performance in the overall system. Each evaluation scenario has been replayed 500 times and the results have been averaged.

8.5.1 Trust Examination

In the following, it is demonstrated how the placement of replicas can be improved — using the proposed algorithm — in response to trust changes in the environment. Therefore, the importance level of services is changed during runtime. We varied the rate of requests for every service and compared the deviation of actual trust from the required trust. Figures 8.4 and 8.5 show the results of this experiment with and without trust, respectively. The results attest a good performance for the trust-based replication compared to the baseline approach. Most of services show either an equal or a better actual trust than the required trust. However, there exists a small number of services which have a less trustworthy state than the required value. This is explained by the fact that nodes in our system have a limited capacity available for storage. This makes the replica process difficult for some services to find still unloaded trustworthy nodes on which to place their replicas.

8.5.2 Overhead Examination

To establish replication, replicas need to be created in the system. In the following, this overhead is investigated for the baseline and the trust-based replication approach. For this experiment, the same settings as above has been used and Figure 8.6 shows its result. The values on the x-axis stand for the replication approach

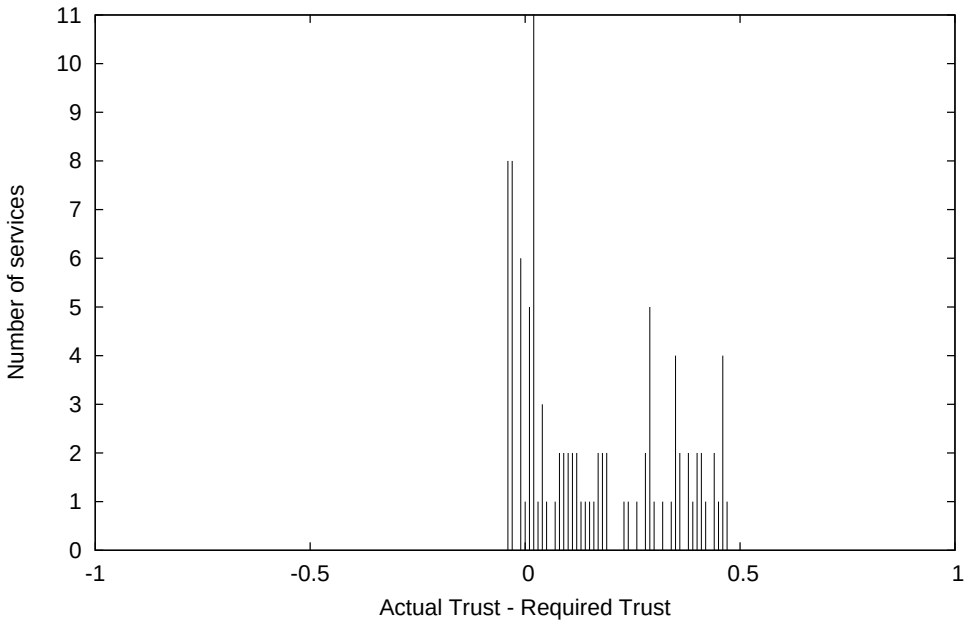


Figure 8.4: The density of trust deviations using the trust-based approach.

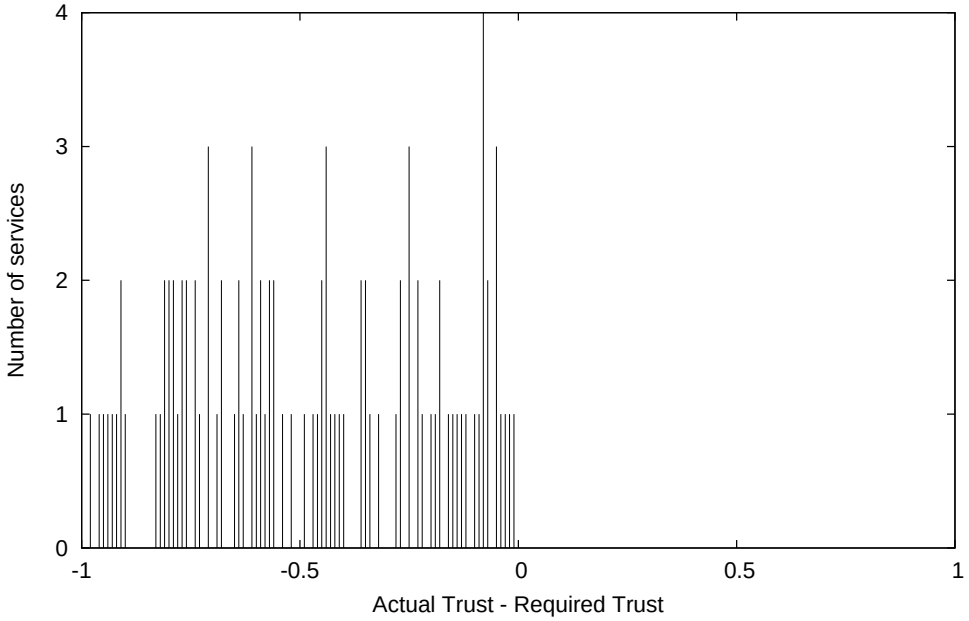


Figure 8.5: The density of trust deviations using the baseline replication approach.

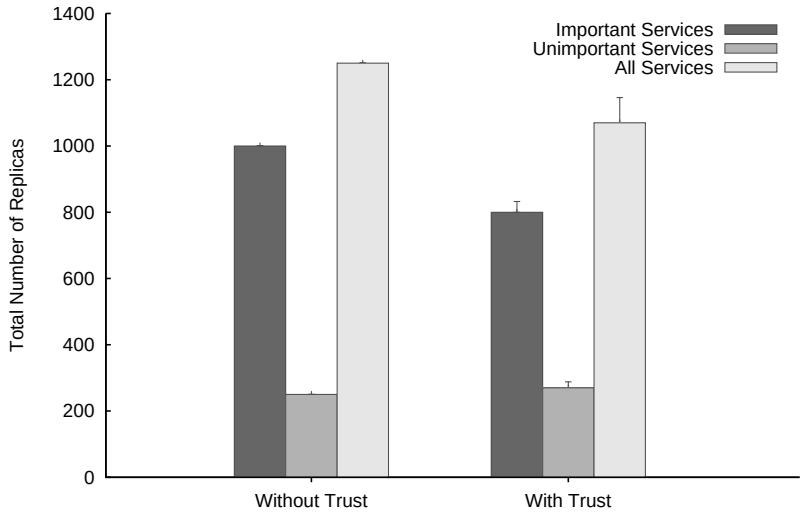


Figure 8.6: Measuring the impact of overhead for different categories of services using the trust-based vs. the baseline replication approach.

used and the total number of replicas is depicted on the y-axis. To perform detailed measurements, we separated the important services from unimportant ones and calculated also the replica overhead created for each category. From the results, we can observe that the overhead is reduced by about 20 % for important services. But for unimportant services, the overhead is maintained nearly at a constant level. This is because the condition set in the system does not tolerate to produce less replicas than R_{min} for unimportant services. Overall, we can say that the cost of replication is decreased using our approach by about 14% for all services. This means that the consideration of trust does not prevent our algorithm to save overhead by placing the replicas.

8.6 Related Work

As far as we know, a previous study of replication for self-healing systems based on trust in open and distributed environments has never been done before. However, some works have already been done on their design, for instance the system designed by Google called Spanner [CDE+13], OriFile [MBHM13], MinCopysets [CSR+13], Scada [KGA+14], CalvinFS [TJA15], etc. Our approach differs from the current sate of the art in four major points. First, the benevolence assumption of nodes is not made in our work. In contrast, we use the social concept of trust to mitigate hazards that can occur from placing replicas randomly in the sys-

tem. Second, our approach can adapt to changing behavior of nodes as well as to changing condition of services. Third, it possesses a regulation mechanism to save replica overhead at runtime, and finally it is applicable to any kind of trust-aware Recommender Systems. Thus, we want to compare our mechanism to systems that employ the same approach. The closest approach to us is to perform the replica placement based on a forecast regarding the availability of nodes. For example, the authors of [LGR15] analyze how to maximize the number of objects that remain available when node failures occur. Existing systems based on such availability placement include [RDB10, MCM15, BMSV03, BLF09]. The main disadvantage of those approaches is that they do not take the priority of services into account and need a high computation power to perform the placement. This would not be suitable for ubiquitous or embedded systems. Very recently, the authors of [HNZSY+16] proposed an interesting replication technique based on trust to minimize the impact of inoperable TMS instances in the system. Based on an artificial exponential distribution to model trust, the authors estimate the trust of each node and determine the amount of replica accordingly. However, as exponential distributions are memoryless they cannot be used to predict trust as we do in this work. The second disadvantage of this approach is that it does not consider the number of requests to determine the importance level of services at runtime.Parts of the content of this chapter have been submitted by the author in the following conference:

- **[MU17a]:** Nizar Msadek and Theo Ungerer. *An Efficient Replication Approach Based on Trust for Distributed Self-Healing Systems*. In ICINCO 2017: Proceedings of the of the 14th International Conference on Informatics in Control, Automation and Robotics, (Submitted but not yet published), Madrid, Spain, Springer 2017.

8.7 Conclusion and Future Work

In this work, a novel replication approach for self-healing systems is proposed. It is based on the notion of trust to improve the trust distribution of replicas and to minimize replication overhead in the system. The algorithm makes use of a trust metric to model the trust relationship between nodes, which is missing in most existing state of the art systems. Then, a mathematical model is formulated to determine the required trust of each service based on how many times it was requested in the last rounds. This is important to enhance the management of replicas for better trustworthiness of important services. An evaluation is provided with respect to a real-world Organic Computing middleware. Overall, the results show a better trust

distribution for replicas with a significant reduction in overhead when compared to the baseline. However, there are still some studies to be done for future work. For instance, further improving the trust distribution of replicas and further decreasing the replication overhead. We also plan to investigate more the categorization of services by including other factors such checkpoint size and service centrality.

9

Conclusions and Future Work

Abstract. The contributions of this thesis are approaches that enable to improve the robustness of self-organizing systems in open and distributed environments. This chapter summarizes these approaches, and gives an overview of the achieved outcomes. Finally, it discusses future research challenges related to the context of the proposed approaches.

9.1 Thesis Summary

This thesis dealt with trust techniques that enable to increase the robustness and performance of self-* systems, in open and distributed environments. After an analysis of the state of the art, a system view has been introduced. This corresponds to the baseline system used for all conducted measurements within this thesis. It belongs to the category of Organic Computing systems and covers the most important characteristics of self-* properties mentioned in the literature. In the light of these properties, a number of key challenges and issues have been identified — to drop the benevolence assumption of self-* systems that hinder their acceptance to be applied in open environments. The investigations conducted in this thesis are solutions for these issues with a strong focus on trust that can be summarized as follows:

Trust-Enhanced Self-Configuration. The self-configuration approach introduced in Section 4.3 is able on the one hand to equally distribute the load of services on nodes as in a typical load balancing scenario and on the other hand to assign services with different importance levels to nodes so that the more important services are assigned to more trustworthy nodes. Evaluations have been conducted to rate the effectiveness of the algorithm when nodes are failing, i.e., the reduction of failures of important services. The results show that the proposed self-configuration algorithm is able to increase the availability of important services by more than 12% compared to the baseline version without trust.

Simultaneous Self-Configuration. The simultaneous algorithm presented in Section 4.5 deals with coordination strategies as enhancement to the aforementioned self-configuration in order to operate with multiple managers at the same time. The results show an outstanding performance for the simultaneous approach with a decrease in processing time of minimum 50%. Furthermore, the simultaneous algorithm includes a fault handling mechanism enabling the system to continue hosting services even in the presence of failures. The types of failures considered in this work are crash failure, execution failure and reachability failure.

Trust-Enhanced Self-Optimization. The introduced self-optimization approach of Section 5.3 is used to optimize the allocation of services on nodes during runtime. It does not only consider pure load-balancing but also takes into account trust to improve the assignment of important services to trustworthy nodes. Different optimization strategies are applied in this context to determine whether a service should be transferred to another node or not. The evaluation results showed that the proposed approach is able to balance the workload between nodes nearly optimal. Moreover, it improves significantly the availability of important services, i.e., the achieved availability was no lower than 85% of the maximum theoretical availability value.

Simultaneous Self-Optimization. A set of variations have been investigated in Section 5.6 for the above mentioned self-optimization in order to improve its performance in case of multiple requests. The difference between the variations arises in the way to handle requests, either sequential or parallel. At the end, a comparative evaluation is conducted to analyze the performance results of the variations compared to the basic approach. The results attest a good performance for the extended optimization algorithm with parallel request handling.

Conflicting Trust Values. The conflict resolution mechanism of Chapter 6 aims

to solve trust conflicts at runtime based on an iterative exchange of trust values between contractors and managers. We analyzed its performance for different scenario cases and the results showed that an average conflict reduction of 97.5%, 53,42% and 6,47% were achieved respectively to the Best-case, Average-case and Worst-case scenario.

Self-Healing: Trust-Based Monitoring. The benefits of the trust-based monitoring algorithm presented in Chapter 7 are twofold. On the one hand, it aims to reduce the number of messages that arises from sending keep-alive messages. On the other hand, it aims to improve the failure detection delays by adapting to the trust conditions of the network. For this purpose, three monitoring functions have been introduced: discrete monitoring, continuous monitoring, and continuous-discrete monitoring. The results show that the continuous-discrete monitoring performs best. It can adapt faster to changing trust condition in the network than the others.

Self-Healing: Trust-Based Replication. The trust-based replication of Chapter 8 is an approach based on trust that provides a good management of replicas, especially for those of important services despite malicious behavior of nodes in the network. Depending on the importance level of a service possessing the replicas and the assessment of the trustworthiness of a node, we can optimize the trust distribution of replicas at runtime. For evaluation proposes we applied our approach to an evaluator based on the TEM middleware. In this testbed, the usage of trust reduced about 14% the replication overhead while providing a much better placement of important replicas than without trust.

9.2 Future Research Challenges

In this section, we discuss future work related to the context of the proposed approaches that can further advance the robustness of self-* systems. Some of these feasible advancements are outlined in the following:

Split-Brain Problem. One limitation of the service recovery that we faced during the evaluation process is the Split-Brain Problem [BK14]. This represents a state in which nodes in the network are partitioned into clusters. And each one believes it is the only active cluster in the network. Figure 9.1 provides a better comprehension of that problem. Assume we have one contractor c_1 that operates to report some service results to manager m. Let us further assume that m can no more communicate with its contractor c_1, due to a reachability failure which can happen at any time in our system. In such case, manager m asks helpers h_1 and h_2 to check

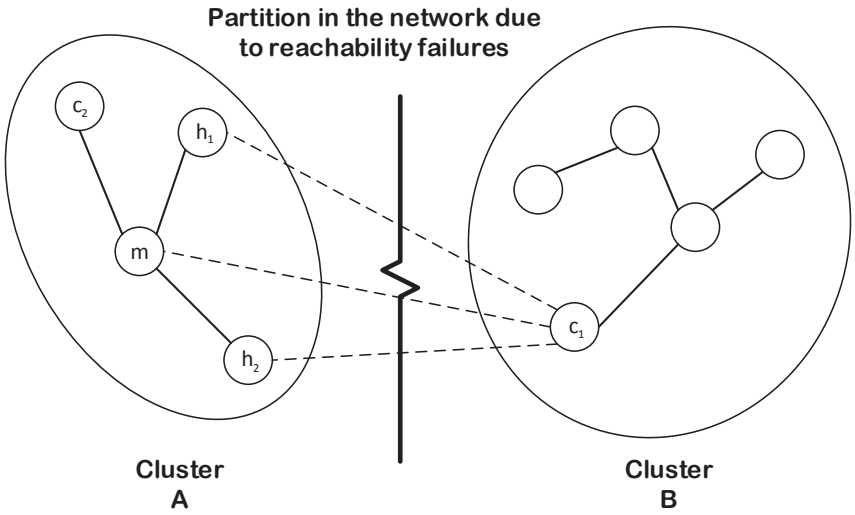


Figure 9.1: Simplified illustration of a network partition isolating a manager m from its contractor c_1 due to a reachability failure in the network. Once the problem is resolved, an automatic reconciliation will be required in order to bring the network in a consistent state again.

whether c_1 is alive or not, as explained above. The two helpers cannot reach c_1. Consecutively, m believes that c_1 has crashed. It uses the data storage to recover all services which were running on c_1 and to restart them on another contractor c_2. Until now, the system seems to run correctly. However, a problem arises when the partition between the two clusters is lost and m is not aware of it, leading to inconsistency in its service results. In the literature, there are many different approaches to deal with the split-brain problem. The most common one is the Quorum approach [Shi06, UKT13], which consists of selecting the cluster with a majority of votes. A disadvantage of this method is that it does not operate if clusters in the system have the same number of votes causing a non-determinism in the solutions computed. Therefore we are interested in providing a better approach that is able to consider additional constraints (beside the voting constraint) such as the last version of services, the number of run services, the workload and trust of contractors and so on. More research related to this is left for future work.

Self-Protecting Mechanisms. For future work, we plan also to investigate more sophisticated self-protecting mechanisms for the TEM that might further increase its robustness against trust manipulation. For this purpose, we have to analyze and study the most common security threats present in the field of trust in distributed

environments such as the ones presented in [MP09] to get a deeper understanding of that issue. Of course, such investigation could also include other related fields such as Cloud Computing and peer-to-peer systems. Based on this study, we have to build self-protecting solutions that can be applied in our trustworthy self-* layer. It is important to note that the found solutions should also be characterized by low overhead in order to be integrated in TEM – no self-protection at any cost.

Extension for Real-Time Systems. Our TEM middleware is currently able to examine the logical correctness of services at runtime. This is performed mainly by comparing the output results of at least two contractors executing the same service, as discussed in Section 3.4.3. However, what the TEM does not provide are temporal correctness mechanisms [WG98] to allow managers in the system to specify which of the services should be well completed in due time, i.e., within a given deadline. Therefore, a step further will be the integration of timing techniques for services, helping the support of real-time applications over TEM. Depending on the importance level of a service, two type of deadlines can be defined [BU10]: namely *firm* and *hard* deadlines. Firm deadlines can be helpful especially for unimportant services. If the deadline is missed, the computation result can loose it relevance but no serious damage will be caused in the system, like a service controlling the GPS position of a car. In contrast, hard deadlines are necessary for important services where the miss may result in harm or damage, such as a service controlling the Airbag system of a car. To guarantee that these deadlines hold, the worst-case execution time (WCET) of every service has to be determined. This corresponds to the the highest possible timing limit that could take a service to be executed over TEM. Interesting will be in this context to investigate different priority inversion mechanisms — as those for instance proposed in [SRL90, HJ06, SMFGG01] — that can be applicable to TEM, and moreover, to find out how the failure detection delay can efficiently be bounded— during the self-healing process — to allow for rescheduling the services within a certain time period set by the deadline. Therefore, it is imperative for the near future to make our trust-enhanced self-* approaches real-time capable. Furthermore, estimating efficiently and precisely the WCET of a service — first with low overhead and foremost without extensive under- and overestimation — is not a simple task to achieve in open systems since this estimation can depend on several uncertain factors, such as service dependencies, resource conflicts, data conflicts, network delays, trust behaviors etc. For this purpose, studies are needed to get a deep understanding of how WCETs can be realistically estimated in open environments and to investigate which state of the art techniques originated maybe from other fields such as Real-Time Systems and Reactive Computing can also be applied to TEM.

On the road to Industry 4.0 & industrial IoT. The developed self-* algorithms provided in this thesis can be seen as a step to further help the realization of smart factories in the current trend of industrial IoT [JBSR17] and Industry 4.0 [PMF+16]. Many autonomous robots — equipped with different computing technologies and tools — could be connected to a network in order to collectively solve services that a single robot cannot solve. The system is open in the sense that robots can enter and leave the network arbitrarily at any time. The behavior of such robots must not influence the correct processing of services, otherwise the business will be exposed to serious risks. The importance level of services can change during runtime. The system should recognize which services are considered to be more important than others and perform the self-configuration process. This self-configuration consists of finding an initial allocation of services allowing the more important services to be operated only from high trustworthy robots. The overall load in the system should be well-balanced as well. If an additional robot joins the network, the system would try to optimize the assignment of services by means of the self-optimization property. The self-healing property comes into play to perceive services that are not operating correctly and to make the necessary adjustments to restore them autonomously, without causing any adverse impact on the system.

Other Possible Future Application Domains. There are a lot of areas in which open self-* algorithms are conceivable to be applied in the technologies that surround us today. Example include Warehousing, health, smart home as well as smart grid.

- **Warehousing:** An intelligent warehouse of the future will include a large number of robots that can move freely among different storage racks within multiple distribution centers. They can shift up and down aisles to store and retrieve cases. They coordinate with other robots autonomously to perform complex tasks, such as optimizing the storage space when facility expansion for instance of one center is no more possible. Self-* algorithm capable of operating in an open environments are highly needed for such a scenario.
- **Health:** Domestic medical robots can be developed in the future to help older people in their home to perform activities such as medication management and to provide services for emergency situations. In such a scenario, there is a need for recognizing important services and balancing their workload between high trustworthy robots, and thus to increase the system's safety and robustness.

- **Smart Home:** Autonomous robots can be tailored in the near future to the individual need of single homes. They provide services increasing the comfort assistance, pleasure, and well-being of residents without restricting their daily routine. Trustworthy self-* algorithms from the basis of realizing such a scenario.
- **Smart grid:** A smart grid of the future will be composed of several heterogeneous entities that cooperate with each other in a parallel way. With the use of an open network the system has to implement robust self-organizing approaches as those introduced in this thesis.

Bibliography

[ABAS16] Abdullah Akbar, S. Mahaboob Basha, and Syed Abdul Sattar. A comparative study on load balancing algorithms for sip servers. *Information Systems Design and Intelligent Applications*, Series Volume 435:79–88, 2016.

[ACE+03] Ralf Allrutz, Clemens Cap, Stefan Eilers, Dietmar Fey, Helmut Haase, Christian Hochberger, Wolfgang Karl, Bernd Kolpatzik, Jochen Krebs, Falk Langhammer, Pawel Lukowicz, Erik Maehle, Jörg Maas, Christian Müller-Schloer, Reinhard Riedl, Burghardt Schallenberger, Volker Schanz, Hartmut Schmeck, Detlef Schmid, Schröder Preikschat, Theo Ungerer, Hans-Otto Veiser, and Lars Wolf. Organic computing - computer- und systemarchitektur im jahr 2010 (in german). *VDE / ITG / GI - Positionspapier*, 2003.

[ADH+08] Tarek Abdelzaher, Yixin Diao, Joseph L. Hellerstein, Chenyang Lu, and Xiaoyun Zhu. Introduction to control theory and its application to computing systems. *Performance Modeling and Engineering*, pages 185 – 215, 2008.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54:2787 – 2805, 2010.

[AÖK10] Ruben Alexandersson, Peter Öhman, and Johan Karlsson. Aspect-oriented implementation of fault tolerance: An assessment of overhead. In *Computer Safety, Reliability, and Security*, 2010.

[ARH97] Alfarez Abdul-Rahman and Stephen Hailes. A distributed trust model. In *Proceedings of the 1997 Workshop on New Security Paradigms*, 1997.

[Art94] W. Brian Arthur. Complexity in economic theory: Inductive

reasoning and bounded rationality. *The American Economic Review*, 84:406–411, 1994.

[ASM+13] Gerrit Anders, Florian Siefert, Nizar Msadek, Rolf Kiefhaber, Oliver Kosak, Wolfgang Reif, and Theo Ungerer. Temas a trust-enabling multi-agent system for open environments. Technical report, Universität Augsburg, 2013.

[ASR15] Gerrit Anders, Florian Siefert, and Wolfgang Reif. A heuristic for constrained set partitioning in the light of heterogeneous objectives. In *Lectures Notes in Artificial Intelligence*, 2015.

[ASS+15] Gerrit Anders, Alexander Schiendorfer, Florian Siefert, Jan-Philipp Steghöfer, and Wolfgang Reif. Cooperative resource allocation in open systems of systems. In *Transactions on Autonomous and Adaptive Systems (TAAS)*, 2015.

[ASTU06] Pietzowski Andreas, Benjamin Satzger, Wolfgang Trumler, and Theo Ungerer. Using positive and negative selection from immunology for detection of anomalies in a self-protecting middleware. *Annual conference of the Gesellschaft für Informatik e.V. (GI),*, 2006.

[AUNDFMGS+15] M. Anis Uddin Nasir, G. De Francisci Morales, D. Garcia-Soriano, N. Kourtellis, and M. Serafini. The power of both choices: Practical load balancing for distributed stream processing engines. *IEEE 31st International Conference on Data Engineering (ICDE), 2015*, pages 137–148, 2015.

[AVFK02] Joakim Aidemark, Jonny Vinter, Peter Folkesson, and Johan Karlsson. Experimental evaluation of time-redundant execution for a brake-by-wire application. In *International Conference on Dependable Systems and Networks*, 2002.

[BA06] Azzedine Boukerche and Kaouther Abrougui. An efficient leader election protocol for mobile networks. In *Proceedings of the 2006 international conference on Wireless communications and mobile computing*, page 6, 2006.

[BCG04] Fabio Bellifemine, Giovanni Caire, and Dominic P. A. Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, Ltd, 2004.

[BEDL13] Olivier Beaumont, Lionel. Eyraud-Dubois, and Hubert. Larchevêque. Reliable service allocation in clouds. In *27th IEEE International Parallel & Distributed Processing Symposium*, 2013.

[BFIK99] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, 1999.

[BIP88] Michael E. Bratman, David Israel, and Martha E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.

[BK14] Peter Bailis and Kyle Kingsbury. The network is reliable: An informal survey of real-world communications failures. In *ACM Journal Queue*, 2014.

[BKHC10] Yvonne Bernard, Lukas Klejnowski, Jö.rg Hähner, and Müller-Schloer Christian. Towards trust in desktop grid systems. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010.

[BKL00] Hamidzadeh Babak, Lau Ying Kit, and David J. Lilja. Dynamic task scheduling using online optimization. In *Journal IEEE Transactions on Parallel and Distributed Systems*, volume Volume 11 Issue 11, 2000.

[BLF09] Samuel Bernard and Fabrice Le Fessant. Optimizing peer-to-peer backup using lifetime estimations. *2nd International Workshop on Data Management in Peer-to-peer systems*, page 8, 2009.

[BMCB05] L.F. Bittencourt, E. R. M. Madeira, F. R. L. Cicerre, and L. E. Buzato. A path clustering heuristic for scheduling task graphs onto a grid. In *In 3rd International Workshop on Middleware for Grid Computing (MGC05)*, 2005.

[BMS02] Martin Bertier, Oliver Marin, and Pierre Sens. Implementation and performance evaluation of an adaptable failure detector. In *International Conference on Dependable Systems and Networks*, 2002.

[BMSV03] Ranjita Bhagwan, David Moore, Stefan Savage, and Geoffrey M. Voelker. Replication strategies for highly available

peer-to-peer storage. *Future Directions in Distributed Computing*, Volume 2584 of the series Lecture Notes in Computer Science:153 – 158, 2003.

[Boz08] Engin Bozdag. A survey of extensions to the contract net protocol. Technical report, Delft University of Technology, 2008.

[BU10] Uwe Brinkschulte and Theo Ungerer, editors. *Mikrocontroller und Mikroprozessoren*. eXamen.press, 2010.

[Bur01] S. Roland Burns. *Advanced Control Engineering*. Butterworth-Heinemann, 2001.

[CDE+13] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google’s globally distributed database. *ACM Trans. Comput. Syst.*, 31:8:1–8:22, 2013.

[CHMS08] Emre Cakar, Jörg Hähner, and Christian Müller-Schloer. Investigation of generic ob-server/controller architectures in a traffic scenario. *Informatik: Beherrschbare Systeme – dank Informatik*, 134:733–738, 2008.

[CSR+13] Asaf Cidon, Ryan Stutsman, Stephen Rumble, Sachin Katti, John Ousterhout, and Mendel Rosenblum. Mincopysets: Derandomizing replication in cloud storage. *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation*, 2013.

[CTA02] Wei Chen, Sam Toueg, and Marcos Kawazoe Aguilera. On the quality of service of failure detectors. In *IEEE Transactions on Computers*, 2002.

[DGB03] Umesh Deshpande, Arobinda Gupta, and Anupam Basu. Performance improvement of the contract net protocol using instance based learning. In *5th International Workshop - Distributed Computing*, 2003.

[DHS07] Ivan Dedinski, Alexander Hofmann, and Bernhard Sick. Cooperative keep-alives: An efficient outage detection algorithm for p2p overlay networks. In *Seventh IEEE International Conference on Peer-to-Peer Computing*, 2007.

[DKRA00] Chrysanthos Dellarocas, Mark Klein, and Juan Antonio Rodriguez-Aguilar. An exception-handling architecture for open electronic marketplaces of contract net software agents (2000). In *Proceedings of the 2nd ACM conference on Electronic commerce*, 2000.

[DL88] Cynthia Dwork and Nancy Lynch. Consensus in the presence of partial synchrony. *Journal of the Association for Computing Machinery*, 35:288–323, 1988.

[DLEJ86] Eager Derek L, D. Lazowska Edward, and Zahorjan John. A comparison of receiver-initiated and sender-initiated adaptive load sharing. In *Conference on Measurement and Modeling of Computer Systems*, 1986.

[DMSGK05] Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos. Self-organization in multi-agent systems. *Knowl. Eng. Rev.*, 20:165–189, 2005.

[DSNH10] Simon Dobson, Roy Sterritt, Paddy Nixon, and Mike Hinchey. Fulfilling the vision of autonomic computing. *Computer*, 43:35 – 41, 2010.

[EAT13] Tomoya Enokido, Ailixier Aikebaier, and Makoto Takizawa. An energy-efficient redundant execution algorithm by terminating meaningless redundant processes. In *International Conference on Advanced Information Networking and Applications*, 2013.

[ESJ+15] Sarah Edenhofer, Christopher Stifter, Uwe Jänen, Jan Kantert, Sven Tomforde, Jörg Hähner, and Christian Müller-Schloer. An accusation-based strategy to handle undesirable behaviour in multi-agent systems. In *International Conference on Autonomic Computing*, 2015.

[FDMD15] Sylvain Frey, Ada Diaconescu, David Menga, and Isabelle Demeure. A generic holonic control architecture for heteroge-

neous multiscale and multiobjective smart microgrids. *ACM Trans. Auton. Adapt. Syst.*, 10:9:1–9:21, 2015.

[FG97] Stan Franklin and Art Graesser. *Intelligent Agents III Agent Theories, Architectures, and Languages: ECAI'96 Workshop (ATAL) Budapest, Hungary, August 12–13, 1996 Proceedings*, chapter Is It an agent, or just a program?: A taxonomy for autonomous agents, pages 21–35. Springer Berlin Heidelberg, 1997.

[FKT01] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 2001.

[GB04] Adriana Giret and Vicente Botti. Holons and agents. *Journal of Intelligent Manufacturing*, 15:645–659, 2004.

[GB12] Eva Geisberger and Manfred Broy. *agendaCPS: Integrierte Forschungsagenda Cyber-Physical Systems*. Springer-Verlag, 2012.

[GBS08] Saurabh Ganeriwal, Laura K. Balzano, and Mani B. Srivastava. Reputation-based framework for high integrity sensor networks. *ACM Trans. Sen. Netw.*, 4:15:1–15:37, 2008.

[GG08] Kunal Goswami and Arobinda Gupta. Resource selection in grids using contract net. In *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 105 – 109, 2008.

[Gla89] Andrew Glassner. *An Introduction to Ray Tracing*. The Morgan Kaufmann Series in Computer Graphics, 1989.

[GMT06] Gabriel Ghinita and Yong Meng Teo. An adaptive stabilization framework for distributed hash tables. In *Proceedings of 21st International Parallel & Distributed Processing Symposium IPDPS*, 2006.

[HAM08] Markus C Huebscher and Julie A. McCann. A survey of autonomic computing - degrees, models, and applications. *ACM Computing Surveys*, 40 Issue 3, 2008.

[HC09] Fu-Shiung Hsieh and Chih Yi Chiang. Workflow planning in holonic manufacturing systems with extended contract net protocol. In *22nd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, 2009.

[HDK03] Naohiro Hayashibara, Xavier Defago, and Takuya Katayama. Two-ways adaptive failure detection with the ϕ-failure detector. *Workshop on Adaptive Distributed Systems*, pages 22–27, 2003.

[HDYK04] Naohiro Hayashibara, Xavier Defago, Rami Yared, and Takuya Katayama. The φ accrual failure detector. *Symposium on Reliable Distributed Systems*, 2004.

[HJ06] Tarek Helmy and Syed S Jafri. Avoidance of priority inversion in real time systems based on resource restoration. *International Journal of Computer Science and Applications*, 3:40 – 50, 2006.

[HNZSY+16] Talal H. Noor, Quan Z. Sheng, Lina Yao, Schahram Dustdar, and Anne H.H. Ngu. Cloudarmor: Supporting reputation-based trust management for cloud services. *IEEE Transactions on Parallel & Distributed Systems*, 27:367 – 380, 2016.

[Hor01] Paul Horn. Autonomic computing: Ibm's perspective on the state of information technology. *IBM Corporation*, pages 1–39, 2001.

[HWA15] Stephan Hammer, Michael Wißner, and Elisabeth André. Trust-based decision-making for smart and adaptive environments. In *User Modeling and User-Adapted Interaction*, 2015.

[HWHMS08] Martin Hoffmann, Michael Wittke, Jörg Hähner, and Christian Müller-Schloer. Spatial partitioning in self-organizing smart camera systems. In *Journal of selected topics in signal processing*, 2008.

[IBM06] IBM. Autonomic computing whitepaper: An architectural blueprint for autonomic computing. Technical report, White Paper, 2006.

[Jam08] M. Jamshidi. System of systems engineering - new challenges for the 21st century. *IEEE Aerospace and Electronic Systems Magazine*, 23:4 – 19, 2008.

[JAR09] Karl Johan Astrom and M. Murray Richard. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton Unversity Press, 2009.

[JBSR17] Sabina Jeschke, Christian Brecher, Houbing Song, and Danda B Rawat. *Industrial Internet of Things*. Cybermanufacturing Systems, 2017.

[JDM03] O.Kephart Jeffrey and Chess David M. The vision of autonomic computing. *IEEE Computer Society*, pages 41 – 50, 2003.

[JMC+06] Branker Jürgen, Mnif Moez, Müller-Schloer Christian, Prothmann Holger, Richter Urban, Rochner Fabian, and Schmeck Hartmut. Organic computing – addressing complexity by controlled self-organization. *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, pages 185 – 191, 2006.

[Jøs96] Audun Jøsang. The right type of trust for distributed systems. In *Proceedings of the 1996 Workshop on New Security Paradigms*, 1996.

[JSS+13] Uwe Jaenen, Henning Spiegelberg, Lars Sommer, Sebastian von Mammen, Jürgen Brehm, and Jörg Hähner. Object tracking as job-scheduling problem. In *Seventh International Conference on Distributed Smart Cameras (ICDSC)*, 2013.

[JSW98] Nicholas R. Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1:7–38, 1998.

[KAA+13] Zuhair Khayyat, Karim Awara, Amani Alonazi, Hani Jamjoom, Dan Williams, and Panos Kalnis. Mizan: a system for dynamic load balancing in large-scale graph processing. *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 169–182, 2013.

[KAS+12] Rolf Kiefhaber, Gerrit Anders, Florian Siefert, Theo Ungerer, and Wolfgang Reif. Confidence as a means to assess the accuracy of trust values. In *The 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2012)*, September 2012.

[KASR15] Oliver Kosak, Gerrit Anders, Florian Siefert, and Wolfgang Reif. An approach to robust resource allocation in large-scale systems of systems. In *Proceedings of the 9th IEEE International*

Conference on Self-Adaptive and Self-Organizing Systems (SASO), 2015.

[KB09] John S. Kinnebrew and Gautam Biswas. Efficient allocation of hierarchically-decomposable tasks in a sensor web contract net. In *Conference on Web Intelligence and Intelligent Agent Technology*, volume Volume 02, pages Pages 225–232, 2009.

[KBA+13] Lukas Klejnowski, Yvonne Bernard, Gerrit Anders, Christian Müller-Schloer, and Wolfgang Reif. Trusted community - a trust-based multi-agent organisation for open systems. In *Proceedings of the 5th International Conference on Agents and Artificial Intelligence (ICAART)*, 2013.

[KGA+14] J. Kirsch, S. Goose, Y. Amir, D. Wei, and P. Skare. Survivable scada via intrusion-tolerant replication. *IEEE Transactions on Smart Grid*, 5:60 – 70, 2014.

[KHS98] Jeffrey O. Kephart, James E. Hanson, and Jakka Sairamesh. Price-war dynamics in a free-market economy of software agents. *Proceedings of the Sixth International Conference on Artificial Life*, 10:53–62, 1998.

[KHS+09] Junichi Kodama, Tomoki Hamagami, Hiroshi Shinji, Takayuki Tanabe, Toshihisa Funabashi, and Hironori Hirata. Multi-agent-based autonomous power distribution network restoration using contract net protocol. In *Electrical Engineering in Japan*, volume 166, 2009.

[KHS+11] Rolf Kiefhaber, Stephan Hammer, Benjamin Savs, Julia Schmitt, Michael Roth, Florian Kluge, Elisabeth André, and Theo Ungerer. The neighbor-trust metric to measure reputation in organic computing systems. In *The 5th IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW 2011)*, pages 41 – 46, October 2011.

[Kie14] Rolf Kiefhaber. *Calculating and Aggregating Direct Trust and Reputation in Organic Computing Systems*. PhD thesis, Augsburg University, 2014.

[KJMU13] Rolf Kiefhaber, Ralf Jahr, Nizar Msadek, and Theo Ungerer. Ranking of direct trust, confidence, and reputation in an abstract system with unreliable components. In *The 10th IEEE In-*

ternational Conference on Autonomic and Trusted Computing (ATC-2013), 2013.

[KSS+10] Rolf Kiefhaber, Benjamin Satzger, Julia Schmitt, Michael Roth, and Theo Ungerer. Trust measurement methods in organic computing systems by direct observation. In *The 8th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC 2010)*, pages 105 –111, December 2010.

[LB11] Melit. Leila and Nadjib Badache. An energy efficient leader election algorithm for mobile ad hoc networks. In *10th International Symposium on Programming and Systems (ISPS)*, 2011.

[LBB15] Andreas Lund, Benjamin Betting, and Uwe Brinkschulte. Design and evaluation of a bio-inspired, distributed middleware for a multiple mixed-core system on chip. In *18th International Symposium on Real-Time Distributed Computing Workshops*, 2015.

[Lee08] E. A. Lee. Cyber physical systems: Design challenges. *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369, 2008.

[LFK+14] Heiner Lasi, Peter Fettke, Hans-georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & Information Systems Engineering*, pages 239–242, 2014.

[LGR15] P. Li, D. Gao, and M. K. Reiter. Replica placement for availability in the worst case. *IEEE 35th International Conference on Distributed Computing Systems (ICDCS)*, 67:599 – 608, 2015.

[LPR+16] Peter R. Lewis, Marco Platzner, Bernhard Rinner, Jim Torresen, and Xin Yao. *Self-aware Computing Systems*. Natural Computing Series, 2016.

[LSJB11] Andreas Lobinger, Szymon Stefanski, Thomas Jansen, and Irina Balan. Coordinating handover parameter optimization and load balancing in lte self-optimizing networks. In *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, 2011.

[Mar94a] Stephen Marsh. *Artificial Social Systems: 4th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW '92 S. Martino al Cimino, Italy, July 29–31, 1992 Selected Papers*, chapter Trust in distributed artificial intelligence, pages 94–112. Springer Berlin Heidelberg, 1994.

[Mar94b] Stephen Paul Marsh. *Formalising trust as a computational concept*. PhD thesis, University of Stirling, 1994.

[Mat89] Friedemann Mattern. Virtual time and global states of distributed systems. In *Parallel and Distributed Algorithms*, 1989.

[MBHM13] Ali José Mashtizadeh, Andrea Bittau, Yifeng Frank Huang, and David Mazières. Replication, history, and grafting in the ori file system. *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ACM:151–166, 2013.

[MCM15] K. Alex Mills, R. Chandrasekaran, and Neeraj Mittal. Algorithms for replica placement in high-availability storage. *The Computing Research Repository (CoRR)*, abs/1503.02654, 2015.

[MD05] Stephen Marsh and Mark R. Dibben. *Trust Management: Third International Conference, iTrust 2005, Paris, France, May 23-26, 2005. Proceedings*, chapter Trust, Untrust, Distrust and Mistrust – An Exploration of the Dark(er) Side, pages 17–33. Springer Berlin Heidelberg, 2005.

[MKFU14] Nizar Msadek, Rolf Kiefhaber, Bernhard Fechner, and Theo Ungerer. Trust-enhanced self-configuration for organic computing systems. In *27th International Conference on Architecture of Computing Systems ARCS2014*, 2014.

[MKU14a] Nizar Msadek, Rolf Kiefhaber, and Theo Ungerer. Simultaneous self-configuration with multiple managers for organic computing systems. In *The 2nd International Workshop on Self-optimisation in Organic and Autonomic Computing Systems (SAOS14) in conjunction with ARCS 2014*, 2014.

[MKU14b] Nizar Msadek, Rolf Kiefhaber, and Theo Ungerer. A trust- and load-based self-optimization algorithm for organic computing systems. In *International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, 2014.

[MKU15a] Nizar Msadek, Rolf Kiefhaber, and Theo Ungerer. A trustworthy, fault-tolerant and scalable self-configuration algorithm for organic computing systems. *Journal of Systems Architecture (JSA)*, Volume 61:Pages 511 – 519, 2015.

[MKU15b] Nizar Msadek, Rolf Kiefhaber, and Theo Ungerer. Trustworthy self-optimization in organic computing environments. *Proceedings of the 28th International Conference on Architecture of Computing Systems*, Series Volume 9017:123–134, 2015.

[MLH+13] Xin Miao, Kebin Liu, Yuan He, Dimitris Papadias, Qiang Ma, and Yunhao Liu. Agnostic diagnosis: Discovering silent failures in wireless sensor networks. In *The IEEE Transactions on Wireless Communications publishes high-quality manuscripts on advances in the state-of-the-art of wireless communications*, 2013.

[MMV03] Vladimír Mařík, Duncan McFarlane, and Paul Valckenaers. *Holonic and Multi-Agent Systems for Manufacturing*. Mařík, Vladimír, 2003.

[MP09] Félix Gómez Mármol and Gregorio Martínez Pérez. Security threats scenarios in trust and reputation models for distributed systems. *Computers & Security*, 28(7):545–556, 2009.

[MS04] C. Müller-Schloer. Organic Computing - On the Feasibility of Controlled Emergence. *CODES + ISSS 2004. International Conference on Hardware/Software Codesign and System Synthesis, 2004.*, 2-5, 2004.

[MS12] Mohammad Amin Morid and Mehdi Shajari. An enhanced e-commerce trust model for community based centralized systems. *Electronic Commerce Research*, 12, 2012.

[Msa15] Nizar Msadek. Trust as a principal ingredient to improve the robustness of self-organizing systems. In *Organic Computing: Doctoral Dissertation Colloquium*, 2015.

[MSKU15] Nizar Msadek, Alex Stegmeier, Rolf Kiefhaber, and Theo Ungerer. A mechanism for minimizing trust conflicts in organic computing systems. In *International Workshop on Self-optimisation in Organic and Autonomic Computing Systems (SAOS)*, 2015.

[MU16a] Nizar Msadek and Theo Ungerer. Trust as important factor for building robust self-x systems. In *Trustworthy Open Self-Organising Systems*, 2016.

[MU16b] Nizar Msadek and Theo Ungerer. Trust-based monitoring for self-healing of distributed real-time systems. In *The 7th IEEE Workshop on Self-Organizing Real-Time Systems (SORT16) in conjunction with ISORC 2016*, 2016.

[MU17a] Nizar Msadek and Theo Ungerer. An efficient replication approach based on trust for distributed self-healing systems. In *Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2017) (submitted but not yet published)*, 2017.

[MU17b] Nizar Msadek and Theo Ungerer. Trustworthy self-optimization for organic computing environments using multiple simultaneous requests. *Journal of Systems Architecture (JSA)*, Volume 75:Pages 26 –34, 2017.

[Mül08] T. Müller. Trusted-computing-systeme. In *Trusted Computing Systeme: Konzepte und Anforderungen*, 2008.

[MWG15] Daosheng Mu, Haoming Wang, and Lijuan Gao. A dynamic adaptive failure detection algorithm based on grey system. In *International Conference on Information Sciences, Machinery, Materials and Energy (ICISMME 2015)*, 2015.

[NB07] Manuel Nickschas and Uwe Brinkschulte. Using multi-agent principles for implementing an organic real-time middleware. In *International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, 2007.

[NB09a] Manuel Nickschas and Uwe Brinkschulte. Carisma - a service-oriented, real-time organic middleware architecture. *Journal of Software*, 4(7):654–663, 2009.

[NB09b] Manuel Nickschas and Uwe Brinkschulte. Carisma - a service-oriented, real-time organic middleware architecture. *Journal of Software*, 4(7):654–663, 2009.

[Oga10] Katsuhiko Ogata. *Modern Control Engineering*. Prentice Hall, 2010.

[Par97] H. Van Dyke Parunak. Go to the ant: Engineering principles from natural multi-agent systems. *Annals of Operations Research*, 75:69–101, 1997.

[PM15] R. Panwar and B. Mallick. Load balancing in cloud computing using dynamic load management algorithm. *International Conference on Green Computing and Internet of Things (ICGCIoT),*, pages 773–778, 2015.

[PMF+16] Christopher Prinz, Friedrich Morlock, Sebastian Freith, Niklas Kreggenfeld, Dieter Kreimeier, and Kuhlenkötter Bernd. Learning factory modules for smart factories in industrie 4.0. *6th CIRP Conference on Learning Factories*, 54:113 – 118, 2016.

[PTT12] Richard Price, Peter Tino, and Georgios Theodoropoulos. Still alive: Extending keep-alive intervals in p2p overlay networks. In *Journal of Mobile Networks and Applications*, 2012.

[QJDL96] Li Qiaoyun, Li Jiandong, Dai Dawei, and Kang Lishan. An extension of contract net protocol with real time constraints. In *Wuhan University Journal of Natural Sciences*, 1996.

[QZGSHK05] Shelley Q. Zhuang, Dennis Geels, Ion Stoica, and Randy H. Katz. Infocom 2005. 24th annual joint conference of the ieee computer and communications societies. proceedings ieee. In *INFOCOM*, 2005.

[RAS+16] Wolfgang Reif, Gerrit Anders, Hella Seebach, Jan-Philipp Steghöfer, André Elisabeth, Jörg Hähner, Müller-Schloer, Christian, and Theo Ungerer. *Trustworthy Open Self-Organising Systems*. Birkhäuser, 2016.

[Ray05] Michel Raynal. A short introduction to failure detectors for asynchronous distributed systems. In *ACM SIGACT News Distributed Computing Column 17*, 2005.

[RDB10] K. Rzadca, A. Datta, and S. Buchegge. Replica placement in p2p storage: Complexity and game theoretic analyses. *IEEE 30th International Conference on Distributed Computing Systems (ICDCS)*, 67:599 – 609, 2010.

[Rei99] Karl Rüdiger Reischuk. *Komplexitätstheorie: Band 1*. Teubner Verlag, 1999.

[RK86] S Rosenschein and L Kaelbling. The synthesis of digital machines with provable epistemic properties. *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 83–98, 1986.

[RLS+12] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica. Load balancing in structured p2p systems. In *Second International Workshop, IPTPS*, 2012.

[RLSS10] Ragunathan (Raj) Rajkumar, Insup Lee, Lui Sha, and John Stankovic. *Proceedings of the 47th Design Automation Conference*, chapter Cyber-physical Systems: The Next Computing Revolution, pages 731–736. ACM Digital Library, 2010.

[RLTB10] Martin Randles, David Lamb, and A Taleb-Bendiab. A comparative study into distributed load balancing algorithms for cloud computing,. In *Advanced Information Networking and Applications Workshops (WAINA)*, 2010.

[RMB+06] Urban Richter, Moez Mnif, Jürgen Branke, Christian Müller-Schloer, and Hartmut Schmeck. Towards a generic observer/-controller architecture for organic computing. In *GI Jahrestagung (1)*, pages 112–119, 2006.

[Rob09] Brunelli Roberto. *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley, 2009.

[RSK+11] Michael Roth, Julia Schmitt, Rolf Kiefhaber, Florian Kluge, and Theo Ungerer. Organic computing middleware for ubiquitous environments. In *Organic Computing - A Paradigm Shift for Complex Systems*, pages 339–351. Springer Basel, 2011.

[RTBL08] Martin Randles, A. Taleb-Bendiab, and David Lamb. Cross layer dynamics in self-organising service oriented architectures. *Self-Organizing Systems*, 5343:293–298, 2008.

[RZSL06] Paul Resnick, Richard Zeckhauser, John Swanson, and Kate Lockwood. The value of reputation on ebay: A controlled experiment. *Experimental Economics*, 9:79–101, 2006.

[SASR13] Jan-Philipp Steghöfer, Gerrit Anders, Florian Siefert, and Wolfgang Reif. A system of systems approach to the evolutionary transformation of power management systems. In *Proceedings of INFORMATIK 2013 – Workshop on Smart Grids*, 2013.

[Sat08] B. Satzger. *Self-healing distributed systems*. PhD thesis, Universität Augsburg, Germany, 2008.

[Sch05] H. Schmeck. Organic computing - a new vision for distributed embedded systems. *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing ISORC*, 42:201–203, 2005.

[SENR13] Jan-Philipp Steghöfer, Benedikt Eberhardinger, Florian Nafz, and Wolfgang Reif. Synthesis of observers for autonomic evolutionary systems from requirements models. In *Workshop on Distributed Autonomous Network Management Systems*, 2013.

[SFSR94] Michael Schillo, Petra Funk, Im Stadtwald, and Michael Rovatsos. Using trust for detecting deceitful agents in artificial societies. *Proceedings of the Ibero-American Conference on Artificial Intelligence (IBERAMIA '94)*, 14:825 – 848, 1994.

[Shi06] K.W. Shirriff. Method and system for establishing a quorum for a geographically distributed cluster of computers. *Google Patents*, US Patent 7,016,946(US7016946 B2), 2006.

[SKB13] Oliver Sander, Alexander Klimm, and Jürgen Becker. Hardware support for authentication in cyber physical systems. *Information Technology*, 55:19–25, 2013.

[SKC15] Hajar Siar, Kourosh Kiani, and Anthony T. Chronopoulos. An effective game theoretic static load balancing applied to distributed computing. *Journal on Cluster Computing*, Volume 18, Issue 4:1609–1623, 2015.

[SKL+10] Jan-Philipp Steghöfer, Rolf Kiefhaber, Karin Leichtenstern, Yvonne Bernard, Lukas Klejnowski, Wolfgang Reif, Theo Ungerer, Elisabeth André, Jörg Hähner, and Christian Müller-Schloer. Trustworthy organic computing systems: Challenges and perspectives. *Proceedings of the 7th International Conference on Autonomic and Trusted Computing (ATC 2010), Springer*, 14:62–76, 2010.

[SLB09] Yoav Shoham and Kevin Leyton-Brown. *MULTIAGENT SYSTEMS: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[SMB+09] Benjamin Satzger, Florian Mutschelknaus, Faruk Bagci, Florian Kluge, and Theo Ungerer. Towards trustworthy self-optimization for distributed systems. In *Software Technologies*

for Embedded and Ubiquitous Systems Lecture Notes in Computer Science Volume 5860, 2009, pp 58-68, 2009.

[SMFGG01] Douglas C. Schmidt, Sumedh Mungee, Sergio Flores-Gaitan, and Aniruddha Gokhale. Software architectures for reducing priority inversion and non-determinism in real-time object request brokers. *Real-Time Systems*, 21:77 – 125, 2001.

[Smi80] Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. In *Defence Research Establishment Atlantic*, pages 1–10. IEEE TRANSACTIONS ON COMPUTERS,, 1980.

[SMR+12] Lionel Seinturier, Philippe Merle, Romain Rouvoy, Daniel Romero, Valerio Schiavoni, and Jean-Bernard Stefani. A component-based middleware platform for reconfigurable service-oriented architectures. In *Software: Practice and Experience*, 2012.

[SMSÇ+11] Hartmut Schmeck, Christian Müller-Schloer, Emre Çakar, Moez Mnif, and Urban Richter. *Adaptivity and Self-organisation in Organic Computing Systems*, chapter Chapter 1.1, pages 5–37. Springer Basel, 2011.

[SPTU08] Benjamin Satzger, Andreas Pietzowski, Wolfgang Trumler, and Theo Ungerer. A lazy monitoring approach for heartbeat-style failure detectors. *International Conference on Availability, Reliability and Security (ARES)*, pages 404–409, 2008.

[SRK+11] Julia Schmitt, Michael Roth, Rolf Kiefhaber, Florian Kluge, and Theo Ungerer. Realizing self-x properties by an automated planner. *International Conference on Autonomic Computing*, 2011.

[SRL90] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers*, 39:1175 – 1185, 1990.

[SU08] Benjamin Satzger and Theo Ungerer. Grouping algorithms for scalable self-monitoring distributed systems. In *International Conference on Autonomic Computing and Communication Systems*, 2008.

[TB06] G. Theodorakopoulos and J. S. Baras. On trust models and trust evaluation metrics for ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 24:318–328, 2006.

[TEP+07] Wolfgang Trumler, Jörg Ehrig, Andreas Pietzowski, Benjamin Satzger, Theo Wolfgang Trumler Ungerer, Robert Klaus, and Theo Ungerer. A distributed self-healing data store. *The 4th International Conference on Autonomic and Trusted Computing*, 2007.

[THW02] H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260–274, 2002.

[TJA15] Alexander Thomson and Daniel J. Abadi. Calvinfs: Consistent wan replication and scalable metadata management for distributed file systems. *13th USENIX Conference on File and Storage Technologies (FAST 15)*, USENIX Association:1 –14, 2015.

[TJHH13] Sven Tomforde, Uwe Jänen, Jörg Hähner, and Martin Hoffmann. Cloud services – towards an intelligent cloud-based surveillance system. In *Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics*, 2013.

[TKU06] Wolfgang Trumler, Robert Klaus, and Theo Ungerer. Self-configuration via cooperative social behavior. In *Autonomic and Trusted Computing (ATC)*, pages 90–99. Springer-Verlag Berlin Heidelberg, 2006.

[TPSU07] W. Trumler, A. Pietzowski, B. Satzger, and T. Ungerer. Adaptive self-optimization in distributed dynamic environments. *First International Conference on Self-Adaptive and Self-Organizing Systems (SASO), 2007*, pages 320 –323, Jul. 2007.

[TSG+15] Alejandro Z Tomsic, Pierre Sens, Joao Garcia, Luciana Arantes, and Julien Sopena. 2w-fd: A failure detector algorithm with qos. In *The 29th International Parallel and Distributed Processing Symposium (IPDPS2015)*, 2015.

[TVS07] Andrew S Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms*. Prentice-Hall, 2007.

[UKT13] Yuuki Ueda, Hideharu Kojima, and Tatsuhiro Tsuchiya. On the availability of replicated data managed by hierarchical voting. In *3 International Conference on Information Science and Cloud Computing Companion*, 2013.

[UMJ+06] Richter Urban, Mnif Moez, Brankergen Jürgen, Mülller-Schloer Christian, and Schmeck Hartmut. Towards a generic observer/controller architecture for organic computing. *Bonner Küllen Verlag*, pages 112–119, 2006.

[VCPG06] Garth V. Crosby, Niki Pissinou, and James Gadze. A framework for trust-based cluster head election in wireless sensor networks dssns. In *Workshop on Dependability and Security in Sensor Networks and Systems*, 2006.

[WG98] Joachim Wegener and Matthias Grochtmann. Verifying timing constraints of real-time systems by means of evolutionary testing. *Real-Time Systems*, 15:275 – 298, 1998.

[WHKA14] Michael Wißner, Stephan Hammer, Ekatarina Kurdyukova, and Elisabeth André. Trust-based decision-making for the adaptation of public displays in changing social contexts. In *Journal of Trust Management*, 2014.

[WK02] Hakim Weatherspoon and John D. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. *Peer-to-Peer Systems*, 2429 of the series Lecture Notes in Computer Science:328 – 337, 2002.

[WYC15] Zhu Wei, Zhuang Yi, and Xu Chaoqun. A double-layer heartbeat detection algorithm orienting to embedded heterogeneous cluster. In *International Journal of Grid Distribution Computing*, 2015.

[WZL+14] K. Wang, X. Zhou, T. Li, D. Zhao, M. Lang, and I. Raicu. Optimizing load balancing and data-locality with data-aware scheduling. *2014 IEEE International Conference on Big Data*, pages 119–128, 2014.

[ZZ10] Besma Zeddini and Mahdi Zargayouna. Multidimentional self-organization for online time-constrained vehicle routing problems. In *4th International KES Symposium on Agents and Multi-agent Systems : Technologies and Applications*, 2010.

List of Figures

List of Tables