1

Advances in Intelligent Systems: Reviews

Sergey Y. Yurish
Editor

IFSA

Advances in Intelligent Systems: Reviews

Book Series, Volume 1

S.Yurish
Editor

Advances in Intelligent Systems: Reviews

Book Series, Volume 1

International Frequency Sensor Association Publishing

S. Yurish, *Editor*
Advances in Intelligent Systems: Reviews, Book Series, Vol. 1

Published by IFSA Publishing, S. L., 2017
E-mail (for print book orders and customer service enquires):
ifsa.books@sensorsportal.com

Visit our Home Page on http://www.sensorsportal.com

e-ISBN: 978-84-697-8923-0
ISBN: 978-84-697-8924-7
BN-20171226-XX
BIC: UYD

Contents

Contents **5**
Preface **11**
Contributors **15**

1. Parallel Optimization for Intelligent Systems: Principles and New Results **19**
1.1. Introduction 19
1.2. Related Work 20
1.3. Basic Idea of the Self-Optimization Algorithm 21
1.3.1. No Optimization *22*
1.3.2. Load Optimization *22*
1.3.3. Trust Optimization *22*
1.3.4. Trust and Load Optimization *23*
1.4. Metrics and Notions 23
1.5. The Algorithm in Detail 25
1.5.1. No Optimization *26*
1.5.2. Load Optimization *26*
1.5.3. Trust Optimization *27*
1.5.4. Trust and Load Optimization *27*
1.6. Multiple Simultaneous Requests 28
1.6.1. Selective Request Handling *29*
1.6.2. Parallel Request Handling *30*
1.7. Evaluation 34
1.7.1. Results Regarding the Rating Function $F_{workload}$ *35*
1.7.2. Results Regarding the Rating Function F_{trust} *36*
1.7.3. Basic Algorithm vs. Extensions *37*
1.7.4. Different Network Settings *39*
1.8. Conclusions and Future Work 44
References 45

2. Task Mapping in Heterogeneous NoC by Means of Population-Based Incremental Learning **47**
2.1. Introduction 47
2.2. The Population-Based Incremental Learning (PBIL) Algorithm 52
2.3. Experimental Results 60
2.4. Concluding Remarks 64
Acknowledgements 64
References 65

3. From Static to Dynamic: A New Methodology for Development of Simulation Applications **69**
3.1. Introduction 69
3.2. Methodology of Dynamic Simulation 74
3.3. Underground Mine Ventilation Systems as Objects of Dynamic Simulation 77

3.3.1. General Overview *77*
3.3.2. Exemplary Models *80*
3.4. Implementation with PHANTOM Framework 83
3.5. Conclusions 86
Acknowledgements 87
References 87

4. Improvement of the Calibration Uncertainty for Class E_1 Weights Using an Adaptive Subdivision Method on an Automatic Mass Comparator 89
4.1. Introduction 89
4.2. Equipment and Standards used in Calibration 91
4.2.1. Some Aspects Regarding the Kilogram "Ni81" *92*
4.2.2. The Weights Involved in Calibration *93*
4.2.3. Precision (or Imprecision) of the Balance *93*
4.3. Mass Determination of Reference Disc Weights Used as Check Standards in the Calibration of E_1 Weights 94
4.3.1. Measurement Matrix Design *94*
4.3.2. Estimated Mass Values for Disc Weights *96*
4.4. Statistical Tools for Evaluation of the Measurement Process and Mass Determination of Class E_1 Weights 97
4.4.1. Method Used to Evaluate the Efficiency of the Weighing Design *97*
4.4.2. Mass Results Obtained in the Calibration of Weights *101*
4.5. Analysis of Uncertainties 103
4.5.1. Uncertainty of the Weighing Process, u_A *103*
4.5.2. Type B Uncertainty *103*
4.5.3. Combined Standard Uncertainty, u_c *104*
4.5.4. Expanded Uncertainty *104*
4.6. Quality Assessment of the Calibration 104
4.7. Conclusions 108
References 108

5. RH Control Developments for Applied Uncertainty Management in Industrial Processes 111
5.1. Introduction 111
5.2. Safety and Security 112
5.2.1. Safety and Security Technologies *113*
5.2.2. Emergency Shut Down Systems *114*
5.2.3. Fire and Gas (F&G) Detection and Alarm Systems *115*
5.2.4. Burner Management Systems *115*
5.3. RH Control the New Level of Decision 116
5.3.1. Generalities *116*
5.3.2. RH Control – the New Challenge *118*
5.3.3. Control and Strategy *120*
5.3.4. Reusability *122*
5.3.5. Software Architecture *122*
5.4. Emerging Technologies 123

5.4.1. Simulation Technologies ... *123*
5.4.2. Developed Computer Networks ... *123*
5.4.3. Intelligent I/O Interfaces ... *123*
5.4.4. High Speed Simulators ... *124*
5.4.5. The Systems Modeling and Simulating with Discrete or Hybrid Events *124*
5.4.6. On Line Testing and Diagnosis ... *124*
5.4.7. Asset Management ... *125*
5.5. System Development Methodologies and Techniques ... 125
5.5.1. System Engineering ... *125*
5.5.2. Extended V Model for Uncertainty Control Development ... *128*
5.5.3. Architecting Systems ... *130*
5.6. Concurrent Engineering ... 131
5.7. Applying Uncertainty Management Principles for System Architecture Design ... 137
5.7.1. A Framework for Control System Architecture Design ... *137*
5.7.2. A Holonic Architecture for Plant Wide Control ... *138*
5.7.3. Integrated Architecture for Real-time Control and Uncertainty Management ... *139*
5.8. Results and Discussion ... 142
5.8.1. Evaluation of the Response Time for Uncertainty Control ... *142*
5.8.2. Methods for Response Elaboration ... *143*
5.8.3. Developed Platforms and Case Studies ... *145*
5.9. Conclusions ... 147
References ... 149

6. A Slow-growing Hierarchy of Time-bounded Programs ... 151
6.1. Introduction ... 151
6.2. Basic Instructions and Definition Schemes ... 153
6.2.1. Recursion-free Programs and Class T_0 ... *154*
6.2.2. Safe Recursion and Class T_1 ... *155*
6.3. Computation by Register Machines ... 156
6.4. The Time Hierarchy ... 159
6.5. Extending the Polynomial-Time Hierarchy to Transfinite ... 161
6.5.1. Structured Ordinals and Hierarchies ... *162*
6.5.2. Diagonalization and Transfinite Hierarchy ... *162*
6.6. The Time-Space Hierarchy ... 165
6.6.1. Recursion-free Programs and Class S_0 ... *166*
6.6.2. Safe Recursion and Class S_1 ... *166*
6.7. Conclusions and Further Work ... 169
References ... 170

7. 173Games as Actors: Interaction, Play, Design and Actor Network Theory ... 173
7.1. Introduction ... 173
7.2. Actor Network Theory ... 174
7.2.1. The Traffic Example ... *175*

7.2.2. Translation *176*
7.2.3. Design as Inscription *177*
7.3. Research Methodology 178
7.4. Case: The Game "Quackle" 179
7.4.1. Quackle! Explained *179*
7.4.2. Game Inscription *180*
7.4.3. Translation *181*
7.4.4. What the Game Does *183*
7.5. Playing a Computer Game 186
7.6. Theory of Play and Games 188
7.7. Exergames 191
7.8. Design Implications 195
7.9. A Word on Scripts 198
7.10. Conclusion and Future Work 199
References 200

8. Multiscale Modelling and Simulation of Fiber-Reinforced Plastics Under Impact Loading 203
8.1. Introduction 203
8.2. State-of-the-Art 204
8.3. Methods of Space Discretization 205
8.4. Ballistic Trials 206
8.5. Numerical Simulation 209
8.5.1. Modelling *210*
8.5.2. Simulation Results *210*
8.5.3. Further Validations *213*
8.6. Conclusions 216
References 218

9. How to Improve Driving Perception on an Advanced Dynamic Simulator While Cornering 221
9.1. Introduction 221
9.2. Methods 222
9.2.1. Participants *222*
9.2.2. Experimental Devices *223*
9.2.3. Experimental Scenario *223*
9.2.4. Task *224*
9.2.5. Experimental Design *224*
9.2.6. Data Analysis *226*
9.3. Results 227
9.3.1. Subjective Analysis *227*
9.3.1.1. Simulator Sickness *227*
9.3.1.2. Realism of Vehicle Behaviour *228*
9.3.1.3. Ease of the Task *231*
9.3.2. Objective Analysis *233*
9.3.2.1. Steering Wheel Reversal Rate *233*
9.3.2.2. Lateral Deviation from the Reference Trajectory *236*
9.4. Discussion of Results 238

9.4.1. Motion Gains ... *239*
9.4.1.1. Motion Realism ... *239*
9.4.1.2. Ease of the Task and Objective Variables ... *241*
9.5. Conclusion ... 244
References ... 245

10. Step Climbing Strategy for a Wheelchair ... 249
10.1. Introduction ... 249
10.1.1. Wheelchair ... *249*
10.1.2. Related Research of Wheelchair Step Climbing ... *251*
10.1.3. Purpose of This Chapter ... *252*
10.2. Theoretical Analysis of Step Climbing for a Wheelchair ... 252
10.2.1. Generating the Driving Force to Lift the Front Wheels (Requirement (1)) ... *254*
10.2.2. Avoidance from Tipping over Backward (Requirements (2), (3)) ... *257*
10.2.3. Generating the Driving Force to Lift the Rear Wheels (Requirement (4)) ... *260*
10.2.4. Result of Simulations ... *262*
10.3. Cooperative Step Climbing Using a Wheelchair and a Robot ... 263
10.3.1. Cooperative Step Climbing System ... *264*
10.3.2. Process of Moving Over a Step ... *269*
10.4. Theoretical Analysis of Cooperative Step Climbing Using the Wheelchair and the Robot ... 273
10.4.1. Requirement of the Manipulator Angles to Avoid Collision and to Grasp the Push Handle ... *277*
10.4.2. Requirement to Exert Enough Driving Force on the Ground to Climb a Step ... *279*
10.4.3. Theoretical Analysis of Cooperative Step Climbing Using the Wheelchair and the Robot ... *281*
10.4.4. Simulation ... *283*
10.5. Experiment of Cooperative Step Climbing ... 285
10.6. Conclusions ... 285
Acknowledgments ... 287
References ... 287

Index ... 289

1.

Parallel Optimization for Intelligent Systems: Principles and New Results

Nizar Msadek and Theo Ungerer

1.1. Introduction

Intelligent 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 [1-3], self-optimization [4-6] and self-healing [7, 8]. 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 [9] 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 [10]. 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:

Nizar Msadek
Department of Computer Science, University of Augsburg, Germany

1) 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 1.3 and 1.4),

2) A formal description of the optimization strategies to determine at runtime whether a service should be transferred to another node or not (see Section 1.5), and

3) A set of extensions for the basic algorithm to further improve its performance time in case of multiple simultaneous requests (see Section 1.6).

All aspects are evaluated and discussed with respect to a toolkit based on the TEM [11], a trustenabling middleware for building real-world distributed Organic Computing systems. Section 1.7 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 1.8.

1.2. 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 [12-17]. In most existing algorithms, the consideration of the trustworthiness of nodes has been neglected so far. For instance, the work of Rao et al. [18] 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. [19] 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 [20], 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 equipment. Matrix [21] 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 [22], 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 [23] 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.

1.3. Basic Idea of the Self-Optimization Algorithm

A distributed system consisting of a set of n nodes $N = \{n_1, n_2.., n_n\}$ 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:

1.3.1. 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

1.3.2. 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*.

1.3.3. 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.

1.3.4. 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*.

1.4. 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 1.1).

$$w_i = \sum_{s \in S_i} c_s, \text{ with } 0 \leq w_i \leq C_i^{max} \qquad (1.1)$$

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 S_i into two sets based on their importance levels:

- S_i^{imp}: Set of important services (running on node i), which are necessary for the functionality of the entire system.
- 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 Fig. 1.1: Simple load optimization method in Fig. 1.1, 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 Fig. 1.2).

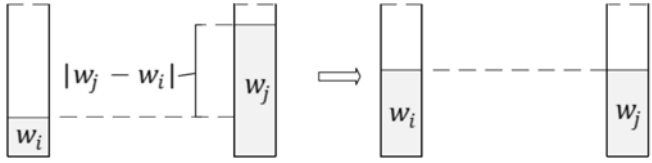

Fig. 1.1. Simple load optimization method.

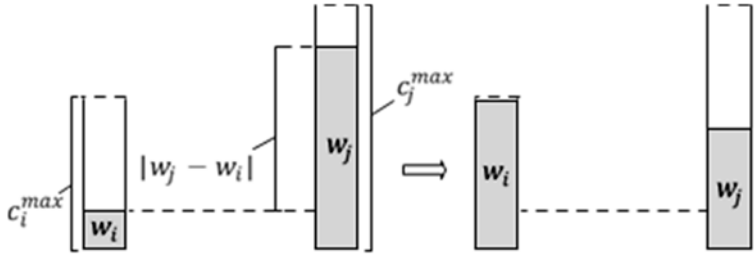

Fig. 1.2. Nodes still unbalanced due to their different resource capacities.

$$Q_i = \frac{w_i + w_j}{c_i^{max} + c_j^{max}} c_i^{max} \quad (1.2)$$

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 1.2. 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 $c^{max}{}_i + c^{max}{}_j$ to obtain the optimal theoretical workload per one unit capacity, which will be then multiplied by $c^{max}{}_i$. Furthermore, each node has an individual trust value calculated based on the previously introduced trust metrics presented in [10]. 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.

1.5. 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 [10] 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:

- S_j^{unimp}: Set of less important services running on node j
- 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

- $c^{max}{}_j$: 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:

1.5.1. 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

1.5.2. 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 1.3 and 1.4) as a set of services that could be selected to balance the workload of nodes. Note that $C(I_s)$ represents the consumption function of a set of services I_s and is calculated by the sum of all its service consumptions.

$$\Psi_{i,j} = \{ I_s | I_s \subseteq \left(S_i^{imp} \cup S_i^{unimp}\right) : maxC(I_s)\ and \qquad (1.3)$$

$$C\ (I_s) \leq \left(O_j - w_j\right)\ and\ 0 < C(I_s) \leq (w_i - O_i)\}$$

$$C(I_s) = \sum_{s \in I_s} C_s \qquad (1.4)$$

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 (1.5.2-a) will be triggered but on the side of j.

1.5.3. 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 1.5) 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 C_{tol} to keep the loadbalancing property in both nodes satisfied.

$$\Psi_{i,j} = \{ I_s | I_s \subseteq S_i^{unimp}, \exists J_s \subseteq S_j^{imp} : maxC(I_s) \; and \qquad (1.5)$$

$$|C(I_s) - C(J_s) \leq C_{tol} \; and \; (C(I_s) + w_i) \leq c_j^{max}\}$$

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 (1.5.4-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 (1.5.3-a), $\Psi_{i,j}$ is determined only from important services (see Formula 1.6), since j is more trustworthy than i. Then, i sends an alert optimization message to j in order to trigger case (1.5.4-a) on side of j.

$$\Psi_{i,j} = \{ I_s | I_s \subseteq S_i^{imp}, \exists J_s \subseteq S_j^{unimp} : maxC(I_s) \; and \qquad (1.6)$$

$$|C(I_s) - C(J_s) \leq C_{tol} \; and \; (C(I_s) + w_i) \leq c_j^{max}\}$$

1.5.4. 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 1.7). Then, *i* transfers $\Psi_{i,j}$ to *j*.

$$\Psi_{i,j} = \{ I_s | I_s \subseteq S_i^{unimp} : maxC(I_s) \quad (1.7)$$

$$and\ C\ (I_s) \leq \left(O_j - w_j\right)\ and\ 0 < C(I_s) \leq (w_i - O_i)\}$$

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 1.8). Then, just as the case of (1.5.4-a), if $\Psi_{i,j}$ is empty, no optimization is done. Otherwise *i* transfers $\Psi_{i,j}$ to *j*.

$$\Psi_{i,j} = \{ I_s | I_s \subseteq S_i^{imp} : maxC(I_s) \quad (1.8)$$

$$and\ C\ (I_s) \leq \left(O_j - w_j\right)\ and\ 0 < C(I_s) \leq (w_i - O_i)\}$$

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 (1.5.4-a or 1.5.4-b) will then be triggered on the side of *j*.

1.6. Multiple Simultaneous Requests

In the evaluation, we have shown that the basic self-optimization algorithm presented in Section 1.5 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. Fig. 1.3 gives an overview of this situation. Let n_i denote the node that receives the requests and let be $L^i = \{l_1, l_2, ..., l_k\}$ 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 $\Psi_{ni,lj}$ for each requester $l_j \in L^i$, using the equations cited in Section 1.5, depending on the current situation of nodes. In the basic approach, as shown in Fig. 1.3, 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.

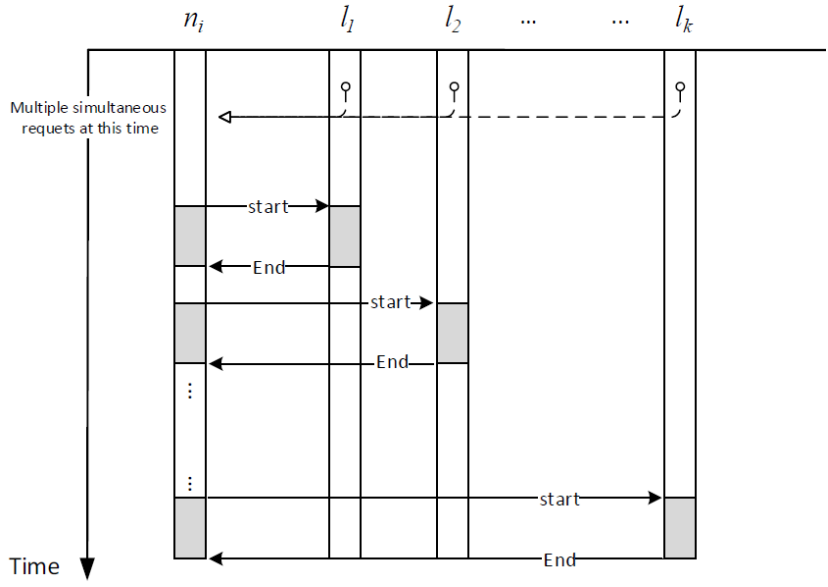

Fig. 1.3. Current execution of the basic algorithm.

1.6.1. 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 set of all involved requesters — in our case always L^i — and S_Ψ is an empty list of fixed size $|L^i|$ 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.

Algorithm 1. Node n_i:

1: $X \leftarrow L^i$ ——→ initialize X as the set of all involved requesters

2: S_Ψ = nil ——→ S_Ψ is initialized as empty list of fixed size $|L^i|$

3: **for** $x \in X$ **do**

4: calculate $|\Psi_{ni,x}|$ 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: $\{x | \exists x \in X : |\Psi_{ni,x}|$ is max and $|\Psi_{ni,x}| > 0\}$

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**

1.6.2. 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 Fig. 1.4. This has the benefit to further decrease the optimization time in the whole system. However,

nodes in our system have different trust and workload 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.

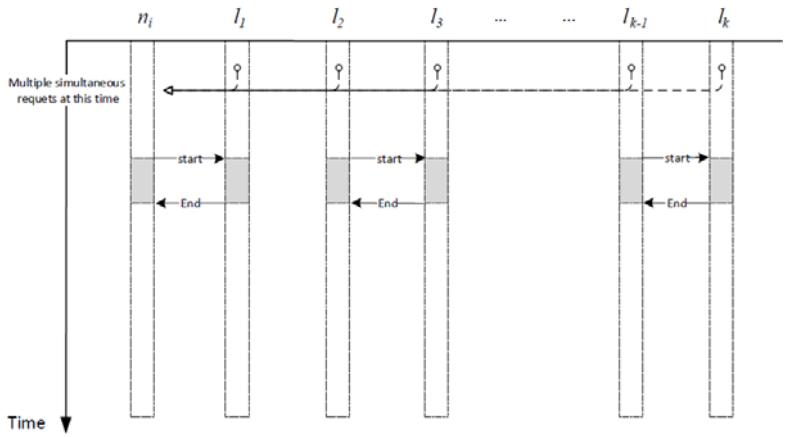

Fig. 1.4. Simplified representation of the parallel request handling.

At the beginning, we initialize two parameters X and T_Ψ. The first parameter $X = \{n_i\} \cup L^i$ 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 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.

Algorithm 2 Node n_i:

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

2: $T_\Psi \leftarrow$ → is an empty lookup table of size $|X| \times |X|$

	n_i	l_1	...	l_k
n_i	0	0	...	0
l_1		0	...	0
...			0	0
l_k				0

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**

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 Fig. 1.5.

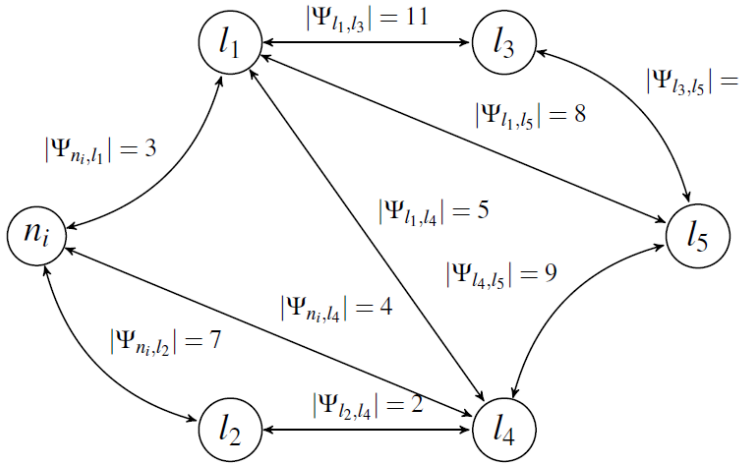

Fig. 1.5. Relation graph of potential service transfers.

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 Fig. 1.6.

$T_\Psi =$

	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						

Fig. 1.6. A simplified representation of T_Ψ after the execution of phase one.

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 $X = \{0/\}$. Hence, the algorithm finishes with the following optimization pairs $\{(l_1, l_3), (l_4, l_5), (n_i, l_2)\}$.

1.7. 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 [11] 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 1.1).

Table 1.1. Mixture of heterogeneous nodes.

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 1.1). If, for example, 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 $F_{workload}$ 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 1.9, where N is the set of all nodes and $|N|$ the cardinality of N. The main idea of the second rating function F_{trust} is to reward important services running on trustworthy nodes. This is expressed by the Formula 1.11, where N is the set of all nodes, 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).

$$Fworkload = \frac{\sum_{n \in N} |workload(n) - \overline{workload}|}{|N|} \tag{1.9}$$

$$\overline{workload} = \frac{\sum_{n \in N} workload(n)}{|N|} \tag{1.10}$$

At the beginning of the simulation, the network is rated by using both F_{trust} and $F_{workload}$. 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., ρ = 50 %. Senders send an application message to receivers to piggyback necessary information for the self-optimization, as described in Section 1.3. 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 F_{trust} should be maximized (i.e., to an optimal theoretical point that we explain later in 1.7.2). 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., $F_{workload}$ should be minimized near to zero.

$$F_{trust} = \sum_{n \in N} \sum_{s \in S_n} p(s)t(n) \tag{1.11}$$

1.7.1. Results Regarding the Rating Function $F_{workload}$

As mentioned above, the first rating function $F_{workload}$ indicates the average workload deviation of all nodes from the desired workload

$\overline{workload}$ (in our case, 50 %). The lower the value of $F_{workload}$, the better the performance of workload balancing.

Fig. 1.7 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.

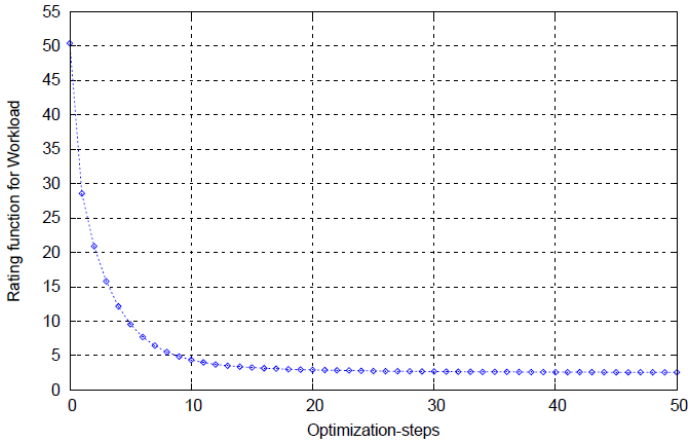

Fig. 1.7. Rating function for the workload deviation ($F_{workload}$).

1.7.2. Results Regarding the Rating Function F_{trust}

In the following, the service distribution for the proposed self-optimization algorithm is evaluated regarding F_{trust}.

Fig. 1.8 shows the result of this experiment. The square line represents the result of F_{trust} using the proposed self-optimization algorithm. It can be observed that the algorithm improves during runtime the availability of important services. 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.

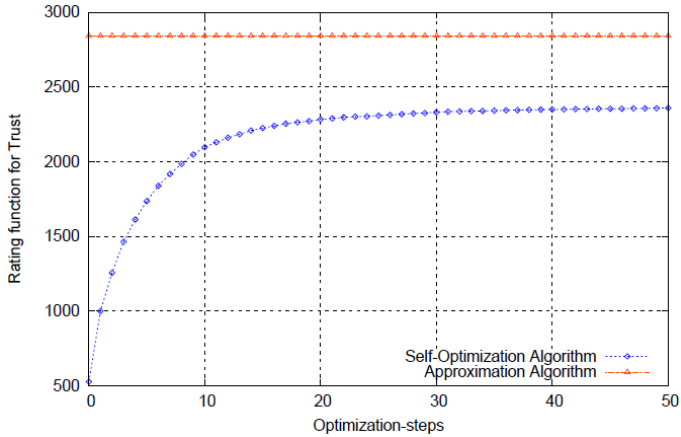

Fig. 1.8. Rating function for Trust (F_{trust}).

1.7.3. Basic Algorithm vs. Extensions

In this section, the gain of applying the proposed extensions with respect to Section 1.6 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 1.6.1).

- **Basic algorithm + Parallel Request Handling (ALG.3):** A variation of the basic optimization algorithm using the extension of the parallel request handling (see Section 1.6.2)

The three algorithms differ in the way they handle multiple requests, either sequential or parallel. Figs. 1.9 and 1.10 present their comparison results with respect to the rating functions F_{trust} and $F_{workload}$.

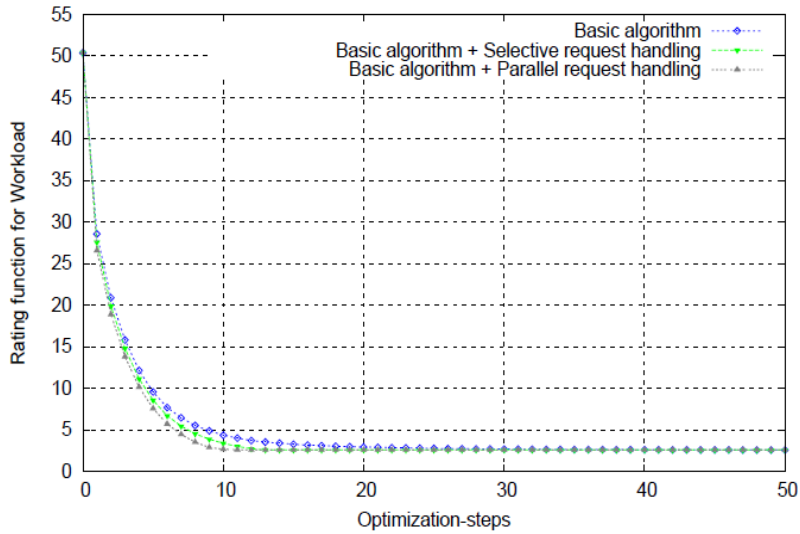

Fig. 1.9. Comparison results according to the rating function $F_{workload}$.

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 everyone 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.

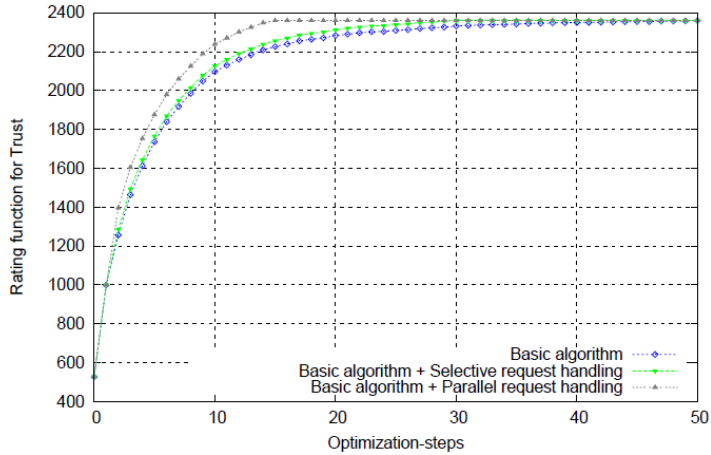

Fig. 1.10. Comparison results according to the rating function F_{trust} .

1.7.4. 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 1.2. Generally, the more trustworthy the nodes are, 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 is set to 45 %. The experiments differ in the adjustment of $|N|$ and ρ. Recall, $|N|$ states for the size of the network and ρ represents 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 $|N| = 100$ but different percentage of ρ.

Table 1.2. 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

• Experiment 1.1: $|N| = 100$, $\rho = 30$ % (see Figs. 1.11 and 1.12)

• Experiment 1.2: $|N| = 100$, $\rho = 50$ % (see Figs. 1.11 and 1.12)

• Experiment 1.3: $|N| = 100$, $\rho = 70$ % (see Figs. 1.11 and 1.12)

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

• Experiment 2.1: $|N| = 200$, $\rho = 30$ % (see Figs. 1.13 and 1.14)

• Experiment 2.2: $|N| = 200$, $\rho = 50$ % (see Figs. 1.13 and 1.14)

• Experiment 2.3: $|N| = 200$, $\rho = 70$ % (see Figs. 1.13 and 1.14)

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

• Experiment 3.1: $|N| = 400$, $\rho = 30$ % (see Figs. 1.15 and 1.16)

• Experiment 3.2: $|N| = 400$, $\rho = 50$ % (see Figs. 1.15 and 1.16)

• Experiment 3.3: $|N| = 400$, $\rho = 70$ % (see Figs. 1.15 and 1.16)

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

• Experiment 4.1: $|N| = 800$, $\rho = 30$ % (see Figs. 1.17 and 1.18)

• Experiment 4.2: $|N| = 800$, $\rho = 50$ % (see Figs. 1.17 and 1.18)

• Experiment 4.3: $|N| = 800$, $\rho = 70$ % (see Figs. 1.17 and 1.18)

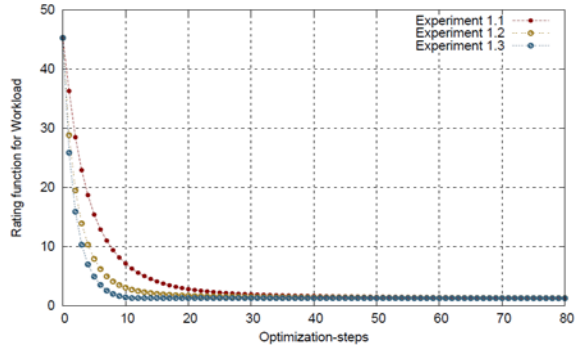

Fig. 1.11. Result of experiments 1.1 - 1-3 according to the rating function $F_{workload}$.

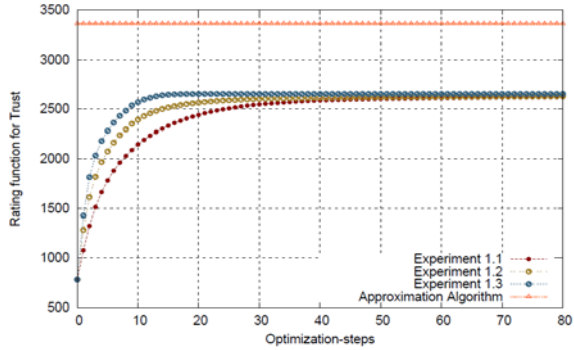

Fig. 1.12. Result of experiments 1.1 - 1-3 according to the rating function F_{trust} .

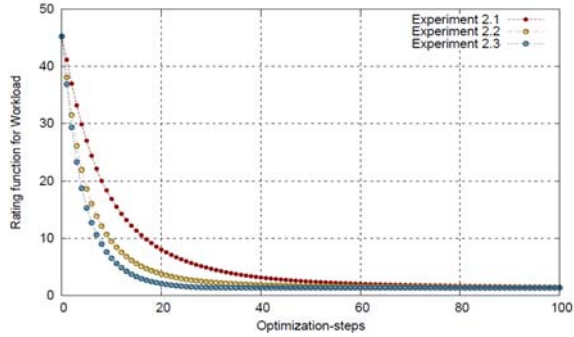

Fig. 1.13. Result of experiments 2.1 - 2-3 according to the rating function $F_{workload}$.

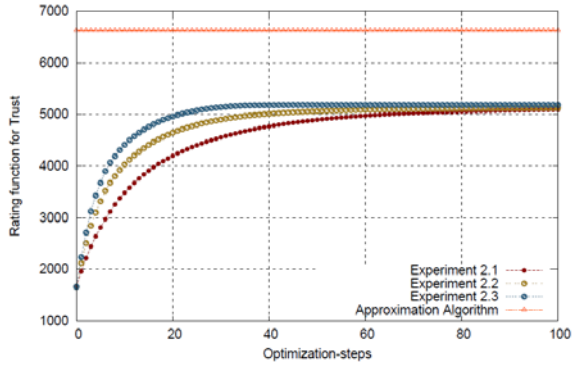

Fig. 1.14. Result of experiments 2.1 - 2-3 according to the rating function F_{trust}.

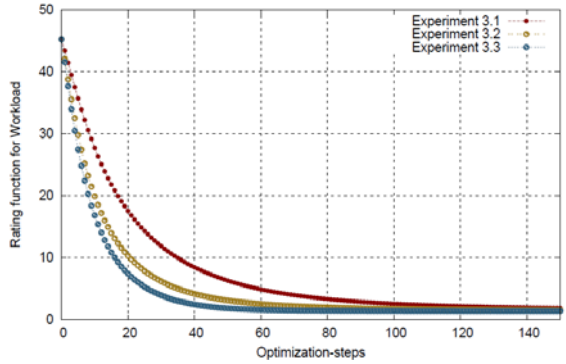

Fig. 1.15. Result of experiments 3.1 - 3-3 according to the rating function $F_{workload}$.

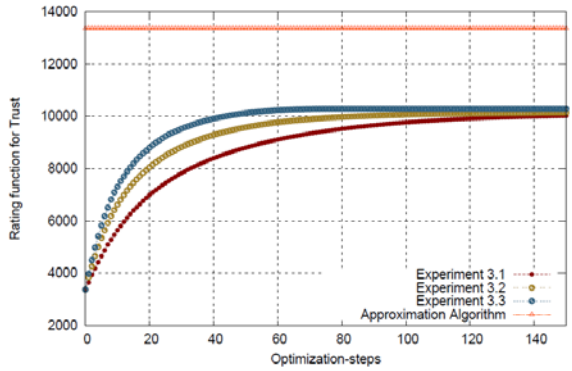

Fig. 1.16. Result of experiments 3.1 - 3-3 according to the rating function F_{trust}.

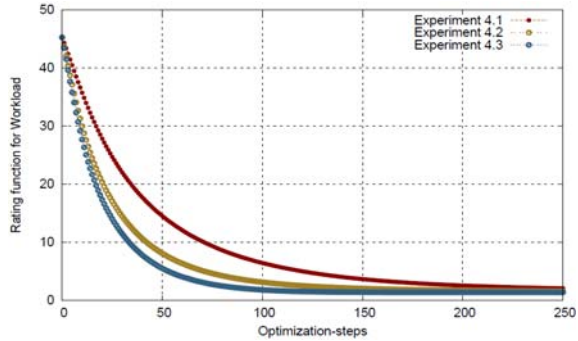

Fig. 1.17. Result of experiments 4.1 - 4-3 according to the rating function $F_{workload}$.

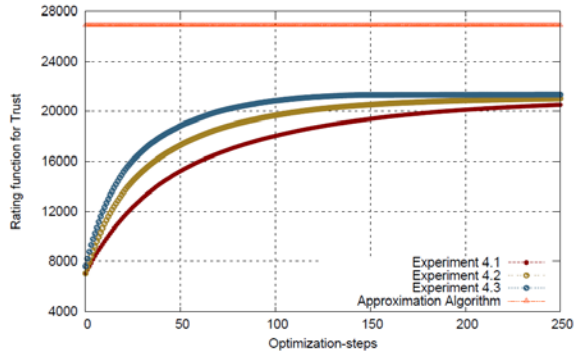

Fig. 1.18. Result of experiments 4.1 - 4-3 according to the rating function F_{trust}

Conclusion Deduced From Conducting Experiments. The experiment results, with the focus on workload, are depicted in Figs. 1.11, 1.13, 1.15, and 1.17. 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. Figs. 1.12, 1.14, 1.16, and 1.18 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.

1.8. 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 1.5 — to determine whether a service should be transferred to another node or not. Section 1.7 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 1.6 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 1.7.3, 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 1.7.4, an additional evaluation 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 from 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 planned 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.

References

[1]. Msadek, N., Kiefhaber, R., Fechner, B., Ungerer, T., Trust-enhanced self-configuration for organic computing systems, in *Proceedings of the 27th International Conference on Architecture of Computing Systems (ARCS2014)*, 2014.

[2]. Msadek, N., Kiefhaber, R., Ungerer, T., Simultaneous self-configuration with multiple managers for organic computing systems, in *Proceedings of the 2nd International Workshop on Self-optimisation in Organic and Autonomic Computing Systems (SAOS'14) in conjunction with ARCS' 14*, 2014.

[3]. Msadek, N., Kiefhaber, R., Ungerer, T., A trustworthy, fault-tolerant and scalable self-configuration algorithm for organic computing systems. *Journal of Systems Architecture (JSA),* Vol. 61, 2015, pp. 511 – 519.

[4]. Msadek, N., Kiefhaber, R., Ungerer, T., Trustworthy self-optimization in organic computing environments, in *Proceedings of the 28th International Conference on Architecture of Computing Systems Series,* Vol. 9017, 2015, pp. 123–134.

[5]. Msadek, N., Kiefhaber, R., Ungerer, T., A trust- and load-based self-optimization algorithm for organic computing systems, in *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO),* 2014.

[6]. Msadek, N., Ungerer, T., Trustworthy self-optimization for organic computing environments using multiple simultaneous requests, *Journal of Systems Architecture (JSA),* Vol. 75, 2017, pp. 26 –34.

[7]. Msadek, N., Ungerer, T., Trust-based monitoring for self-healing of distributed real-time systems, in *Proceedings of the 7th IEEE Workshop on Self-Organizing Real-Time Systems (SORT'16) in conjunction with ISORC'16,* 2016.

[8]. Msadek, N., Ungerer, T., Trust as important factor for building robust self-x systems, in *Proceedings of the Trustworthy Open Self-Organising Systems,* 2016.

[9]. Msadek, N., Trust as a principal ingredient to improve the robustness of self-organizing systems, in *Proceedings of the Organic Computing: Doctoral Dissertation Colloquium,* 2015.

[10]. Kiefhaber, R., Jahr, R., Msadek, N., Ungerer, T., Ranking of direct trust, confidence, and reputation in an abstract system with unreliable components, in *Proceedings of the 10th IEEE International Conference on Autonomic and Trusted Computing (ATC-13),* 2013.

[11]. Anders, G., Siefert, F., Msadek, N., Kiefhaber, R., Kosak, O., Reif, W., Ungerer, T., Temas a trust-enabling multi-agent system for open environments. Technical report, *Universitat Augsburg*, 2013.

[12]. Khayyat, Z., Awara, K., Alonazi, A., Jamjoom, H., Williams, D., Kalnis, P., Mizan: a system for dynamic load balancing in large-scale graph processing, in *Proceedings of the 8th ACM European Conference on Computer Systems,* 2013, pp. 169–182.

[13]. Babak, H., Kit, L. Y., Lilja, D. J., Dynamic task scheduling using online optimization, in *Proceedings of the Journal IEEE Transactions on Parallel and Distributed Systems.,* Vol. 11, Issue 11, 2000.

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

[15]. Siar, H., Kiani, K., Chronopoulos, A. T., An effective game theoretic static load balancing applied to distributed computing, *Journal on Cluster Computing,* Vol. 18, Issue 4, 2015, pp. 1609–1623.

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

[17]. Akbar, A., Basha, S. M., Abdul Sattar, S., A comparative study on load balancing algorithms for sip servers, *Information Systems Design and Intelligent Applications Series,* Vol. 435, 2016, pp. 79–88.

[18]. Rao, A., Lakshminarayanan, K., Surana, S., Karp, R., Stoica, I., Load balancing in structured p2p systems, in *Proceedings of the 2nd International Workshop (IPTPS),* 2012.

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

[20]. Lobinger, A., Stefanski, S., Jansen, T., Balan, I., Coordinating handover parameter optimization and load balancing in the self-optimizing networks, in *Proceedings of the IEEE 73rd Vehicular Technology Conference (VTC Spring),* 2011.

[21]. Wang, K., Zhou, X., Li, T., Zhao, D., Lang, M., Raicu, I., Optimizing load balancing and data-locality with data-aware scheduling, in *Proceedings of the IEEE International Conference on Big Data,* 2014, pp. 119–128.

[22]. Satzger, B., Mutschelknaus, F., Bagci, F., Kluge, F., Ungerer, T., Towards trustworthy self-optimization for distributed systems, in *Proceedings of the Software Technologies for Embedded and Ubiquitous Systems Lecture Notes in Computer Science,* Vol. 5860, 2009, pp. 58-68.

[23]. Derek L, E., Edward, D. L., John, Z., A comparison of receiver-initiated and sender-initiated adaptive load sharing, in *Proceedings of the Conference on Measurement and Modeling of Computer Systems*, 1986.

Advances in Intelligent Systems: Reviews Volume 1

Sergey Y. Yurish, Editor

'Advances in Intelligent Systems: Reviews' Vol. 1 Book Series is covering some design and architectural aspects related to intelligent systems and software. It ranges from the microarchitecture level via the system software level up to the application-specific architecture level.

The book volume contains ten chapters written by 25 contributors from academia and industry from 8 countries: Colombia, Denmark, France, Germany, Italy, Japan, Romania and USA.

The book will be a valuable tool for those who involved in research and development of various intelligent systems.

ISBN 978-84-697-8923-0

IFSA