Grid agent cooperation strategies inspired by Game Theory

Yvonne Bernard, Lukas Klejnowski, Ronald Becher, Markus Thimm, Jörg Hähner, Christian Müller-Schloer
Leibniz Universität Hannover, FG System- und Rechnerarchitektur, Appelstr. 4, D-30167 Hannover

Abstract

The Organic Computing (OC) Initiative deals with technical systems consisting of a large number of distributed and highly interconnected subsystems. In the OC-Trust project we aim at introducing trust mechanisms to improve and assure the interoperability of these subsystems. One application scenario used in our project is a Distributed Desktop Grid where agents act on the behalf of the user and try to distribute work units. In this scenario, we introduce trust-based self-organisation algorithms in order to enhance the efficiency and robustness of such a system. In this paper, we discuss an algorithm for the distributed matchmaking in such a system, which has been inspired by strategies developed in Game Theory.

1 Introduction

Nowadays, most technical systems consist of tightly interconnected, dynamic and complex subsystems. Such complexity can lead to system states that reduce system performance and efficiency. In order to cope with this threat, new approaches in hardware, software and embedded architectures are needed. Therefore, the Organic Computing Initiative [1] introduces concepts to enable subsystems to control certain parts of their functionality autonomously and in a distributed fashion. In our view, Desktop Grids show similar concepts as Organic Computing systems regarding their dynamic and complex structure, local view and adaptation abilities of the entities within the system. Therefore, we enhance Desktop Grid systems with self-organisation algorithms and runtime adaptation. Additionally to the self-organisation mechanisms and agent autonomy, the OC-Trust Research Unit introduces trust as a mechanism to counter the information uncertainty in such dynamic and open systems. The subsystems (or agents) are given a predefined level of autonomy in order to let them adapt to changes in the system or the environment and make decisions runtime.

We assume a Grid and Volunteer Computing System (DGVCS) consisting of agents representing the users of standard computers from different administrative domains. The agents decide to which extend they contribute to the agent community. There is a potential for fraud or at least uncooperative behaviour because agents might try to maximise their benefit and at the same time minimise their effort, i.e. consume much more computing power than they offer to others. In order to cope with these threads, conventional DGVCSs rely on verification methods like check pointing or majority voting which lead to high overhead costs and system workload.

Therefore, we introduce trust as a concept to cope with egoistic behaviour in such systems. Agents rate the result of an interaction and consider the ratings of other agents while deciding with whom to cooperate. Thus, they are able to omit safety measures like majority voting and increase the overhead costs produced by these additional computation steps in the network.

If agents rely on ratings of results of former interactions with a peer, based on either own experience or experience of other agents (reputation), they are able to determine whether cooperation with this peer is worthwhile. Matchmaking algorithms enhanced with the usage of trust values are able to reduce safety measures like work unit replication. This is due to the fact that the verification of results (e.g. majority voting) is not necessary if there is a trust value indicating that the peer will behave cooperatively. In this paper, the trust-based self-organisation algorithms introduced in [2] and [3] are equipped with a new decision mechanism which incorporates strategies well-known from Game Theory. This is done in order to make the agents not only adaptive but also enable them to act strategically based on the expected behaviour of other agents. These strategies will then be enhanced with prediction methods in order to forecast the behaviour of the interaction partner in the next matchmaking step.

The remainder of the extended abstract is organised as follows: In Section 2, we define the system model used in our scenario. In Section 3, we introduce the trust-adaptive agent architecture which is the basis or our implementation. In Section 4, the tactical agent which incorporates the cooperation strategies inspired by Game Theory is introduced. Section 5 extends this model with our novel prediction mechanism for agent behaviour to a learning tactical agent. The evaluation of both approaches is given in Section 6. Section 7 will give a brief overview of related work. In Section 8, we will summarise the work conducted in this paper and give an outlook on future work.

2 System Model

We consider a Desktop Grid where each computer node is represented by an agent. The agent acts on behalf of its user and can access system resources within a user-defined corridor. Agents act in two roles: submitter and

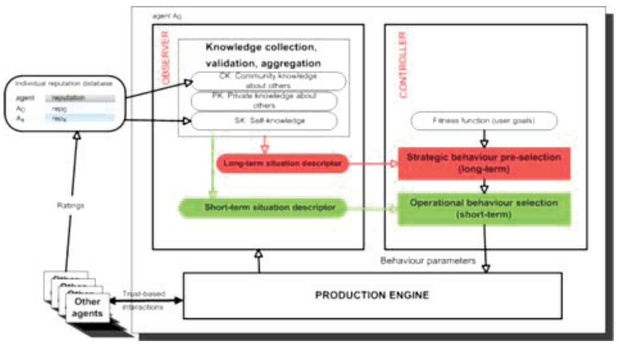
worker. An agent is able to act as a submitter and as a worker at the same time.
As a **submitter**, the agent manages the distribution of jobs which are spitted into work units. The jobs are assumed to be parallelisable, e.g. video rendering or complex face recognition algorithms with large data sets. The number of work units per job and the work unit processing complexity are randomly distributed. The work unit distribution is based on the trust value of possible workers as well as the expected processing time (determined by the possible worker's resources and waiting queue).
In **worker** role, the agent decides which work units to accept for processing by its computer node. In this role, the cooperation strategies inspired by game theory which will be introduced in Section 4 are used to define whether or not to accept the work unit.
Distribution as well as acceptance of work units is done autonomously by the agents. In this early status of our strategy approach, we presume that the local values needed for the strategic system are accessible globally for all agents.
We simulated our Desktop Grid using the agent-based framework RePast [12], which enables us to abstract from conventional communication and neighbourhood considerations for the evaluations presented in this paper.

3 Trust-adaptive agent architecture

In this section, the adaptive agent architecture, which is the basis of our tactical agent implementation, is briefly introduced.
In Subsection 3.1, the generic model of this architecture will be introduced. In Subsection 3.2, we will show how this architecture was used for the implementation of the tactical agent for the desktop grid scenario.

3.1 Generic trust-adaptive agent architecture

The introduction of adaptation allows for an agent to change its behaviour at run-time in reaction to different environmental situation.
For instance, an agent in a DGVCS can choose based on the current situation, whether to accept a work unit which has been offered to it or not. If its current reputation is very low, it needs to improve it in order to be able to act as a submitter in the future. If its reputation is high, it can be worthwhile not to accept the offered work unit and therefore optimise the current fitness.
In order to permit the system designer to steer the autonomy which has been given to the agents, we have introduced the generic agent architecture [3]. This architecture enables agents to dynamically adapt their behaviour at runtime.


agent A0
OBSERVER
Knowledge collection, validation, aggregation
CK: Community knowledge about others
PK: Private knowledge about others
SK: Self-knowledge
Individual reputation database
agent
reputation
A0
rep0
A1
rep1
Long-term situation descriptor
Short-term situation descriptor
CONTROLLER
Fitness function (user goals)
Strategic behaviour pre-selection (long-term)
Operational behaviour selection (short-term)
Ratings
Behaviour parameters
PRODUCTION ENGINE
Trust-based interactions
Other agents
Figure 1. Trust-adaptive agent architecture

As can be seen in Figure 1, the architecture is composed of the following main components:
The *Production engine* is the actual application specific component of the system. We improve the results reached by this Production engine (e.g. a grid client software) by reconfiguring its behaviour at run-time. It can be supplied with optimised parameter sets determined by our higher-level Observer/Controller component.
The *Observer* manages the agent's world model. We differentiate between three different knowledge sources: Knowledge about the agent itself (SK), private knowledge about other agents which has been gathered in former interactions (PK), and community knowledge (CK) about other agents (esp. from the reputation database). The Observer hast to collect, aggregate and validate knowledge from these three categories in order to generate a significant and compact description of the current situation, the situation Descriptor *SD*. This description is presented to the Controller in two abstractions. The long-term Situation Descriptor *SD.L* addresses the expected development of a situation, while the short-term situation Descriptor *SD.S* has a smaller scope but higher level of detail.
The *Controller* uses these *SD*s to determine a suitable behaviour. This decision is made in two steps: First, a long-term (strategic) behaviour pre-selection is chosen which is best suited for the observed long-term situation *SD.L.* For instance, an agent might decide that a more cooperative behaviour is best suited if a decline in its reputation value is diagnosed. This pre-selected behaviour delivers the input for the short-term decision component of the observer: It maps the current short-term situation $SD.S$ with the pre-selected long-term behaviour to a certain current behaviour. The chosen behaviour is represented by a behaviour vector *B* = *<b_{1},..,b_{n}>* which selects a desired behaviour (see Table 1) from the configuration space of the *Production Engine*.

3.2 Trust-adaptive agent architecture for a Desktop Grid

We will now show how the generic architecture introduced above has been applied to our application scenario DGVCS. We are able to present a first approach using a simplified $SD.S$ to show the general concept and feasibility of the idea.
As introduced above, there exist two representations of the agent's situation: *SD.L* for the long-term effects and *SD.S* for the short-term situation description. In the cur-

rent implementation, we concentrated on *SD.S*, hence the current situation of an agent is the foundation of its decisions. In this paper, the agent is able to store its own local interaction results in addition to the information gained from *SD.S*.
The situation description *SD.S* which is delivered by the observer contains the following attributes:

- own benefit: the normalised time the agent has saved by participating in the desktop grid
- average benefit: the normalised time the agents in the trusted community have saved by participating in the desktop grid
- Maximal Average Benefit: The threshold which determines if the agent needs to change between trust Building and Always accept state.
- Own compute/submit threshold: This value determines the whether the agent has already processed enough work units for other agents.
- Maximal compute/submit threshold: The threshold which decides if the commitment of the agent is within the scope desired by the user or the strategic pre-selection component of the controller.

In this paper, we consider two decisions an agent has to take: whom to give work units to process (submitter role) and whether to accept offered work units (worker role).

4 Cooperation strategies inspired by Game Theory

If an agent receives a request to process a work unit for another agent, it has to decide whether or not to accept this request (worker role). Always accepting work units would increase an agent's own workload and therefore decrease the usage of the system for own work units or foreground tasks executed by the user of the PC. Never accepting work units would lead to bad reputation values. This would lead to a situation where the agent is not able to successfully commit a work unit to the system. Therefore, the agent needs to find a trade-off between accepting and rejecting work units. Furthermore, this trade-off is influenced by the current situation the agent acts in, e.g. own workload, overall workload, average overall reputation and own reputation and trust values. We here introduce tactical agents, which apart from the aforementioned values also take into account a prediction of how the other agents might behave in the future.
We now consider a situation where an agent A_i has had a former interaction with an agent A_j.
A_i has stored the result of this former interaction. As can be seen in Table 1, there are five possible results:

- A_j has accepted the work unit offered by A_i
  - and **finished** it correctly.
  - and **canceled** the computation or delivered a wrong result.
- A_j has rejected the work unit offered by A_i due to
  - A_is bad **trust** value (from former interactions of A_j with Ai)
  - A_is bad **reputation**.
  - its own full waiting **queue**.
  - **egoistic** behaviour strategies.

Last Reaction	Accept		Reject			
Behaviour	Finish	Cancel	Trust	Reputation	Queue	Egoism
Always accept	A	A	A	A	A	A
Trust building	A	A	A	A	A	D
Tit for Tat	A	D	D	D	A	D

Table 1. Work unit acceptance behaviours of tactical agent A_i

If A_i is now asked by A_j to process a work unit, it decides depending on the result of the previous interaction and its current strategy. In the basic implementation, the previous interaction is exactly the last one. In Section 5, we will introduce the prediction method we have developed in order to regard not only the last but also a history of past interactions.
As can be depicted from the names, Always accept is a simple strategy showing only one homogeneous behaviour. The well-known Tit for Tat strategy accepts if its last work unit, which has been offered to the requesting agent, has been accepted and successfully calculated, in all other cases it rejects the work unit. In order to improve its reputation value, the agent can follow the trust-building strategy. This strategy is more cooperative than Tit for Tat such that it also accepts if the work unit has been rejected due to understandable reasons, i.e. its own reputation value, a filled waiting queue or a (submission) failure. Only if the rejection was due to pure egoism, the trust-building strategy will reject the work unit.
Figure 1 shows the four strategies an agent can use to decide whether or not to accept a work unit it has been offered.
The basic ideas of these strategies have been adapted from the prisoner's dilemma. There is a variety of possible other strategies, but we started with these well-known ones and will enhance the set of strategies in the future. Agents are able to continuously adapt their strategy to the current situation, e.g. change from Tit for Tat to Trust building if the workload increases or the own reputation value decreases. Figure 1 presents a state machine showing the transitions between strategies.

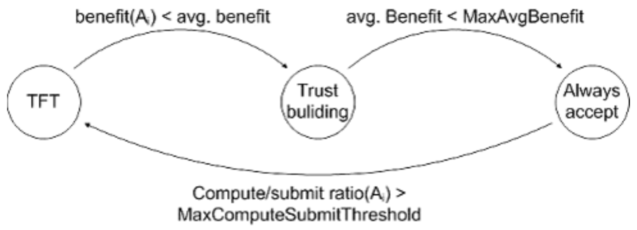
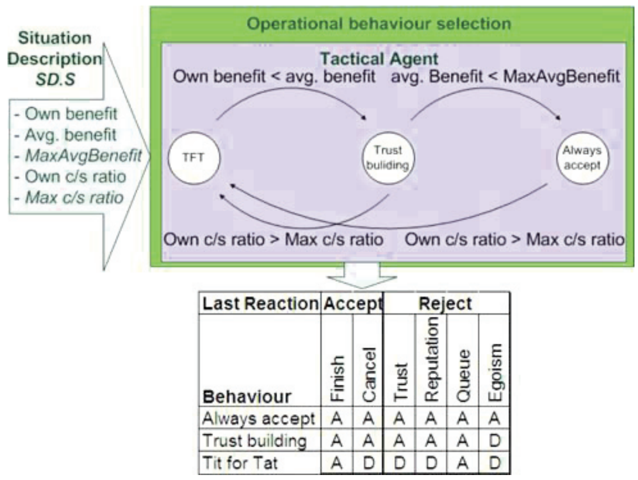

benefit(A_i) < avg. benefit
avg. Benefit < MaxAvgBenefit
TFT
Trust buliding
Always accept
Compute/submit ratio(A_i) > MaxComputeSubmitThreshold

Figure 1. State machine of the tactical agents' behaviour transitions

A tactical agent leaves the Tit for Tat behaviour state if it realises that the average benefit in the Trusted Community is currently higher than its own benefit. This suggests that there is a behaviour allowing for more benefit in the current situation. It will then change to Trust building which is a more cooperative behaviour state. If the agent in this state realises that its benefit is lower than a threshold predefined by the system designer or even the agent's user, it will change to the very cooperative Always accept state. The Always accept state is left if the agent realises that its own commitment (compute/submit ratio) is higher than a predefined threshold. The agent then changes to the least cooperative Tit for Tat state.
In Figure 2, the complete behaviour selection process is given: The tactical agent is a possible component of the operational behaviour selection component (cf. Section 3).


Situation Description SD.S
- Own benefit
- Avg. benefit
- MaxAvgBenefit
- Own c/s ratio
- Max c/s ratio
Operational behaviour selection
Tactical Agent
Own benefit < avg. benefit
avg. Benefit < MaxAvgBenefit
TFT
Trust building
Always accept
Own c/s ratio > Max c/s ratio
Own c/s ratio > Max c/s ratio

Last Reaction	Accept		Reject			
Behaviour	Finish	Cancel	Trust	Reputation	Queue	Egoism
Always accept	A	A	A	A	A	A
Trust building	A	A	A	A	A	D
Tit for Tat	A	D	D	D	A	D

Figure 2. Tactical agent as an operational behaviour selection component

The operational behaviour selection component receives the *SD.S* as introduced in Section 3. Using the Tactical agent selection method, the situation description leads to a state in the state machine which then defines the behaviour given in Table 1. With each new situation description being delivered, the current state is adapted at runtime.

5 Prediction of agent cooperation based on former interactions

In order to take into account more than one past interaction for the cooperation decision, we enhanced our tactical agents with a history data structure storing the results of former interactions with all other agents.
We then implemented a simple prediction method based on double exponential smoothing. This prediction was used as an input for the novel prediction method based on neural networks. The results stored in the history data structure were transferred into a numeric value which was then used as input for the neural network.

5.1 Basic behaviour prediction using double exponential smoothing

Our first prediction approach is based on time series analysis and is used for a short-term prediction of the cooperation partner's next behaviour. This approach is used for the initialisation of the training data set of the neural network introduced in Subsection 5.2.

In the formulae above and below, x is the value of the time series where x' is the value of the simple exponential smoothing function and x'' is the value of the double exponential smoothing. x* denotes the values of the prediction by the exponential smoothing function used.
Here, α (0<= α <=1) is a weight which determines to which extend previous values are taken into account: A small α weights new values much higher than previous ones. Thus, the time series is only smoothed to a small extend and an adaptation is done very fast. A high α weights old values higher, new values are only accounted for to a small extend. This leads to a much smoothed time series and thus to a very slow adaptation to changes in the situation. In this paper, we have chosen α=0.5 which results in a medium adaptive and smooth function and led to the best experimental results.
We use double exponential smoothing as shown in Formula 1.

$$x'' = \alpha x'_t + (1-\alpha) x''_{t-1}$$

Formula 1. Double exponential smoothing

To compute this value, we need to determine the simple exponential smoothing function with prediction given in Formula 2.

$$x^*_{t+1} = x'_t = \alpha x_t + (1-\alpha) x'_{t-1} = \alpha x_t + (1-\alpha) x^*_t$$

Formula 2. Exponential smoothing

The reason of using double exponential smoothing is that, if the data implies a trend, this function is able to recognise this trend and thus predict the next value in the time series.

$$x^*_{t+1} = 2x'_t - x''_{t-1} = x'_t + (x'_t - x''_{t-1})$$

Formula 3. Prediction using second order exponential smoothing

Formula 3 now shows the actual prediction function of the double exponential smoothing used in this paper.
As stated above, we use this prediction to determine the cooperation probability of the possible cooperation partner in the next interaction time step.

5.2 Advanced behaviour prediction using artificial neural networks

The neural network used in our approach has a three layer architecture as shown simplified in Figure 3.
The first layer consists of input nodes, the second layer of hidden nodes and the third layer of output nodes. The neural network used in this paper consisted of 30 input nodes, 100 intermediate nodes and one output node.

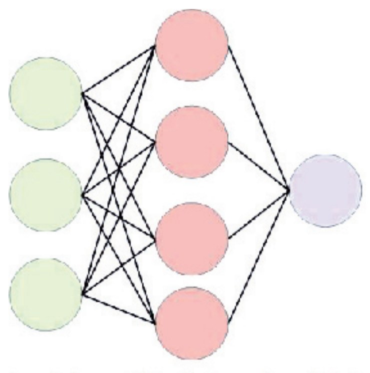


Figure 3. Schematic view of Neural Network for behaviour prediction

Each input nodes is connected to the hidden nodes via a weighted edge. Analogously, the intermediate nodes are connected to all output nodes by a weighted edge.
The weights of the edges are determined in the training phase preceding the usage of the neural network. The data set of inputs and matching outputs has been constructed using the double exponential smoothing prediction in our system. Thus, the weights have continuously been adapted by comparing the output generated by the neural network with the output data perceived by the training set. If the error between these two values is above a given threshold, the weights are repeatedly adapted using resilient propagation (cf. [4]).
After this training phase, the neural network is able to react not only to situations learned by now, but will also find a suited output for new, unknown situations. Additionally, the neural network will optimise the weights and thus the output of its calculation continuously at runtime.

The inputs of our neural network are the previous behaviours of the agent requesting cooperation (cf. Table 1). As we need numerical values as inputs, we transferred these behaviours introduced in Section 4 into double values which best matched the impact of the behaviour to the agent itself (cf. Table 2).

BEHAVIOUR	Value
ACCEPT AND FINISH	1.0
REJECT DUE TO QUEUE	0.8
REJECT DUE TO C/S-DIFFERENCE	0.8
REJECT DUE TO TRUST	0.6
NO ENCOUNTER YET	0.5
REJECT DUE TO REPUTATION	0.4
ACCEPT BUT ABORT	0.2
REJECT DUE TO EGOISM	0.0

Table 2. Agent behaviour transferred into numerical values

For instance, if agent A_j has accepted and successfully calculated a work unit for agent A_i, this positive behaviour has been rated with a 1.0. If it has accepted, but aborted the work unit, this is a very serious misbehaviour because A_i has already waited for the completion of the work unit, but now needs to find another suited worker and has to wait for the second completion attempt. Thus, we rate this behaviour with a 0.2, which is very near to the worst behaviour (reject due to egoism) rated with 0.0. These ratings have been determined by preceding experiments, but other ratings and rankings of the fatality of the misbehaviour are conceivable.

In contrast to the approach introduced in Section 4, we here do not only consider the last interaction but a set of interactions. Thus, the input to our input nodes consists of a vector of n interaction result double values. In the neural network, this vector is reduced to a single double value. Then, we add a random value between 0.0 and 0.2.

This addition led to better results in the experiments, as it led to a scattering of the chosen agent: it reduces the probability that an agent only cooperates with the same agent. Furthermore, this enables agents with a former behaviour which was not entirely cooperative to get a work unit accepted by other agents and thus eventually re-enter the Trusted Community [2], i.e. the set of agents which cooperate well with each other due to their trust values, over time.

The result at the output nodes is a double value r ($0 <= r <= 1$). It represents the cooperation probability of the agent in the next interaction. This r is now used by the agent as an acceptance probability: Instead of delivering A or D (Accept or Decline), as done by the state machine given in Section 4, we will now return A with a probability of r and D with a probability of $1-r$. Thus, we use the expected behaviour of the requesting agent A_j as our chosen behaviour to answer the question whether or not to accept the work unit offered by A_j.

In contrast to the approach given in Section 4, we are now able to decide on a continuous cooperation probability parameter scale rather than a discrete one.

6 Evaluation

In Subsection 6.1, we have compared the time, which the agent has saved by distributing work units instead of calculation them on its own, of agents having rational and tactical behaviour strategies.
Rational agents accept work units until a predefined compute/submit threshold is reached. In this experiment, we have chosen the tactical agents according to the state diagram given in Section 4.

In order to improve the benefit, we decided to enhance our tactical agents with a history of many former interactions combined with a prediction method which have been introduced in Section 5. The results are shown in 6.2.

6.1 Performance improvement using Tactical agents

The first experiment has been conducted in order to determine the success of tactical agent behaviour as introduced in Section 4. Figure 4 shows an example in which we measured a network of 200 agents for 300000 Ticks. The agents in this network had to cope with a high workload which in our system was realised by a job generation probability uniformly distributed between 300 and 1000 ticks. This workload results in queue sizes which are drawn in green (maximal queue size) and red (average queue size), but will not discussed further at this point. The *time save* is a measure how much calculation time (in ticks) an agent has saved by participating in the network during the simulation. The brighter magenta colored points show the time save of the tactical agent (TS TB) whereas the blue points shows the time saved by agents showing a rational behaviour. Here, rational behaviour means an agent accepts Work Units offered by other agents as long as its Work Unit queue is empty enough, and rejects if the queue is full. It can be seen that tactical agents show a slightly higher benefit than rational agents. This is even though there are no completely malicious agents in the network, but only agents showing egoistic behaviour if they are in an egoistic mode of their state machine introduced in Figure 1. Thus, using tactical agents which store the last result of an interaction with an agent would lead to a slightly faster calculation of work units and thus a slightly higher throughput in the network without noticeable calculation or storage overhead.
As the time save is an increasing value, we chose a first simple *fitness function* in order to determine the benefit of a strategy defined as

$$Fitness = (t_{own} - t_{actual}) / t_{ideal}$$

where t_{own} is the time it would have taken the agent itself to process a work unit, t_{actual} is the time it has actually taken to calculate the work unit in the Grid and t_{ideal} the time it would have taken to process the work unit in a perfect system with enough resources for fast and parallel processing. In this paper, the scale of this fitness is between 0 and 25 with 25 being the best value.

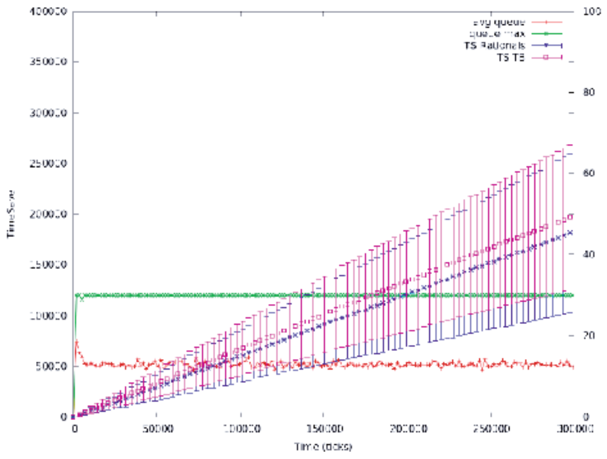


Figure 4. Time saves of rational and tactical agents under high workload

Figure 5 shows the results of a system with 100 tactical agents according to Section 4 under low workload conditions aggregated by the Performance Levels of the agents. The Performance Level PL is a simplified measure for the different system configurations and capacities (e.g. CPU, RAM, HD, network resources). For instance, an agent with PL 4 can process a work unit twice as fast as an agent with PL 2. In this experiment, the lower PLs 2 and 3 reach a higher fitness and thus benefit more from the system than more performing agents. This is due to the fact that they would have needed more time to calculate the work units on their own, slow machines.

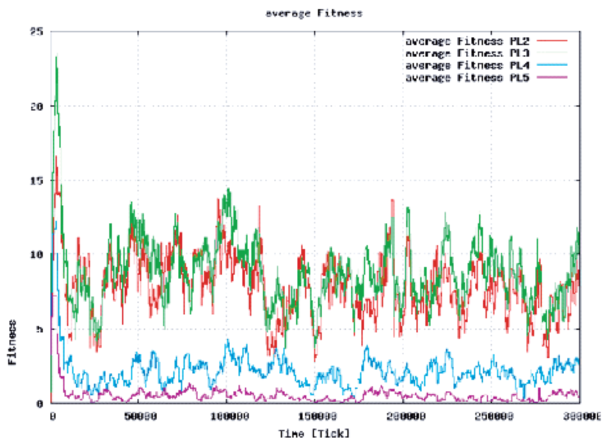


Figure 5. Fitness of tactical agents (Section 4)

In order to further improve this performance increase, we introduced the prediction of agent behaviour based on neural networks. Our first results of this approach are introduced in 6.2.

6.2 Tactical agents enhanced with behaviour prediction

The experiment shown in Figure 6 has again been conducted with 100 tactical agents under low workload conditions. In this experiment, the tactical agents have been improved by the prediction method introduced in Section 5.2.

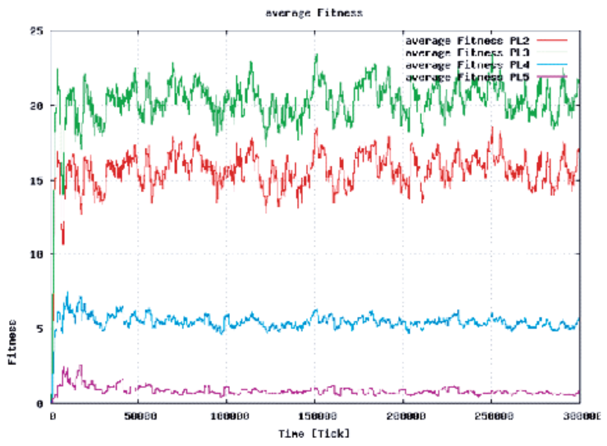


Figure 6. Fitness of tactical agents with neural network prediction (Section 5.2)

Comparing Figures 5 and 6, it can clearly be seen that the prediction based on neural networks leads to performance improvement under the same conditions as the simple tactical agent.
These first promising results hint that using tactical autonomous agents inspired by Game Theory for the matchmaking of Grid tasks can lead to performance improvement and thus a faster calculation of users' work units.

7 Related work

Our application scenario DGVCS [13] is a dynamic, large-scale system where agents act as workers and submitters.
Much research has been done in the area of P2P-based desktop grid systems, e.g. [5] and [6]. However, most of the systems which have been investigated in literature focus on clients from the same administrative domain or a specific (e.g. scientific) community [10]. Here, trust is not as important as in the kind of systems we explore, having no common community goal but rather using idle computing power in order to locally reach faster calculation results. Research on trust-enhanced desktop grid systems can among others for instance be found in [11]. The authors have introduced the EigenTrust model for P2P-based systems (e.g. the Gnutella protocol) which is based on the notion of transitive trust. In our model, trust cannot be regarded as a transitive function in all cases.
Our adaptivity approach is even more far-reaching, as it does not use the trust model but the matchmaking and cooperation algorithms as well as the agent architecture to ensure adaptivity.
In accordance with [9], we regard using agents as an autonomous representation of the user in a grid system as a worthwhile approach.
In our self-organised approach, we combine trust-enhancement with agent adaptivity within designer-defined bounds. In this paper, we extended our agents from the already novel adaptive [3] to even tactical behaviour. The tactical behaviour is seen with respect to results of former interactions and inspired by mechanisms from Game Theory [8]. The agents are taking into account not only trust and reputation values but also a prediction of the future behaviour of possible cooperation partners. The forecast is based on an artificial neural network [7] which processes the history of former interactions.

8 Conclusion and Future work

In this paper, we have introduced the concept of tactical agent whose behaviour determination is inspired by Game Theory.
After briefly introducing our system model and the trust-adaptive agent architecture, which is the basis or our implementation, we discussed the tactical agent which incorporates the cooperation strategies inspired by Game Theory. We then extended this model with our novel prediction mechanism for agent behaviour based on artificial neural networks. The evaluation of both approaches has shown that the tactical agent leads to a small performance improvement, the tactical agent with prediction led to an even better fitness.
In the future, we are going to extend the knowledge of our agents by refining the prediction methods and using them for further aspects. By now, the prediction can be seen as an expected degree of cooperation of a possible cooperation partner. This is a promising first approach, but many other prediction and trend analysis for the Observer of our adaptive agent are possible. For instance, the agents could forecast an increasing workload and proactively adjust their cooperation and distribution parameter in order to quickly adapt their behaviour to the changed situation.
Additionally, the fitness function will be replaced by a function weighting both benefit (e.g. scaled time saved by distribution) and the effort undertaken in order to reach this benefit.

References

[1] Müller-Schloer, Schmeck, "Organic Computing - Quo Vadis?", Chapter 6.2 Organic Computing - A Paradigm Shift for Complex Systems, Birkhäuser, 2011, ISBN 978-3034801294

[2] Bernard, Klejnowski, Hähner, Müller-Schloer, "Towards Trust in Desktop Grid Systems", ccgrid, pp.637-642, 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010

[3] Klejnowski, Bernard, Hähner and Müller-Schloer, "An architecture for trust-adaptive agents", Proceedings of the 2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop (SASOW 2010)

[4] M. Riedmiller and Heinrich Braun, „Rprop - A Fast Adaptive Learning Algorithm.", Proceedings of the International Symposium on Computer and Information Science VII, 1992

[5] J. B. Ernst-Desmulier, J. Bourgeois, M. T. Ngo, F. Spies, and J. Verbeke,"Simulating and optimizing a peer-to-peer computing framework," in Proc. 20th International Parallel and Distributed Processing Symposium IPDPS 2006

[6] Anglano, M. Canonico, M. Guazzone, M. Botta, S. Rabellino,S. Arena, and G. Girardi, "Peer-to-peer desktop grids in the real world: The sharegrid project," Cluster Computing and the Grid, IEEE

[7] P.Hamilton, "Künstliche neuronale Netze: Grundprinzipien, Hintergründe, Anwendungen", Berlin/Offenback, vde-Verlag 1993, ISBN 3-8007-1824-3

[8] M. J. Osborne, "An introduction to game theory", New York: Oxford University Press, 2003.

[9] I. T. Foster, N. R. Jennings and C. Kesselman,"Brain Meets Brawn: Why Grid and Agents Need Each Other", Towards the Learning Grid, IOS Press, 2005, pp. 28-40.

[10] A. J. Chakravarti, G. Baumgartner, and M. Lauria, "Application-specific scheduling for the organic grid," in GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing. Washington, DC, USA: IEEE Computer Society, 2004, pp. 146–155.

[11] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in WWW '03: Proceedings of the 12th international conference on World Wide Web. New York, NY, USA: ACM, 2003, pp. 640–651.

[12] Repast - Recursive Porous Agent Simulation Toolkit.[Online]. Available: http://repast.sourceforge.net/

[13] S. Choi, H. Kim, E. Byun, M. Baik, S. Kim, C. Park, and C. Hwang, "Characterizing and classifying desktop grid," in Cluster Computing and the Grid, 2007. CCGRID 2007. 7th IEEE International Symposium on, 2007, pp. 743 –748.