Self-Organized Task Allocation for Computing Systems with Reconfigurable Components

Daniel Merkle, Martin Middendorf, Alexander Scheidler
Parallel Computing and Complex Systems Group
Department of Computer Science, University of Leipzig
Augustusplatz 10/11, D-04109 Leipzig, Germany
{merkle, middendorf, scheidler}@informatik.uni-leipzig.de

***Abstract*— A self-organized allocation scheme for service tasks in computing systems is proposed in this paper. Usually components of a computing system need some service from time to time in order perform their work efficiently. In adaptive computing systems the components and the necessary tasks adapt to the needs of users or the environment. Since in such cases the type of service tasks will often change it is attractive to use reconfigurable hardware to perform the service tasks. The studied system consists of normal worker components and helper components which have reconfigurable hardware and can perform different service tasks. The speed with which a service tasks is executed by a helper depends on its actual configuration. Different strategies for the helpers to decide about service task acceptance and reconfiguration are proposed. These strategies are inspired by stimulus-threshold models that are used to explain task allocation in social insects.**

I. Introduction

The computing systems that are addressed in this paper follow the paradigm of autonomic computing (e.g., [12]) or organic computing (e.g., [11]). Such computing systems consist of many relatively independent components and they are adaptive to the needs of the user or the environment. Certain management tasks should preferably be done by the systems themselves and they should have the so called self-x properties, i.e., they should be self-configurable, self-optimizing, and self-servicing. The integration of self-organization principles is an important design topic for theses systems. Service tasks for example might be executed by helper components that self-organize the allocation of their service work.

The principles that are used by social insects to organize the work within a colony are potentially interesting for the design of organic computing systems because insect colony organisation usually works without complicated communication mechanisms and without central control. Stimulus-threshold models are one standard type of models that are used in the literature to explain the self-organized labour division in social insect colonies (see, e.g., [2], [4], [17]). In these models each individual has for each task a personal threshold value which determines the preference of this individual to work for that task. In addition, for every task there exists a stimulus value which determines how necessary it is that individuals work for the task. The probability that an individual works for a task depends on the relative size of its threshold value for the task and the stimulus value of the task. The lower the threshold value and the higher the stimulus value the more likely is it that an individual works for the task.

In the first stimulus-threshold models that have been proposed in the literature the fixed threshold values were used ([4]). These fixed threshold models have then been extended so that the threshold of an individual for a task can change dynamically. In the so called threshold reinforcement models it is assumed that the threshold of an individual for a task decreases when the individual works for the task and vice versa it increases when the individual does not work for the task ([2], [17]). Thus, individuals can specialize to certain tasks. The influence of colony size on the degree of specialization of the individuals has been studied in [10], [15]. So far stimulus-threshold models have been applied in scheduling [5], [6], [7], [8], [13], robotics [1], [14], for mail retrieval problems [3], [16], and in multi agent systems [9].

In this paper we propose a self-organized scheme to assign service tasks in computing systems. Two types of components are distinguished here to model the computing systems - the normal workers and the helpers which fulfill servicing tasks. Each worker needs some servicing from time to time before it can continue its normal work. If a worker needs servicing it searches for a helper that offers a suitable servicing task. There exist different types of servicing tasks. The helpers are assumed to have reconfigurable hardware that can perform the different service tasks. The speed with which a servicing task can be performed by a helper depends on how much of its reconfigurable resources are configured for this type of servicing task. Each helper that gets a request for service from a worker decides whether it accepts the request. If it accepts the request it decides whether it reconfigures itself in order to increase the amount of computing resources that are configured for the requested type of service. If the request for service it not accepted the worker searches for another helper.

In this initial study on the topic we start with a simple scenario where the workers might need only two different types of service. The influence of the communication costs and the reconfiguration costs on the performance of the system are studied analytically and experimentally. Different scenarios with static or dynamically changing composition of the servicing tasks are considered. We compare the self-organized tasks allocation scheme with a fixed allocation scheme where each helper task always accepts a request and

1-4244-0054-6/06/$20.00 ©2006 IEEE

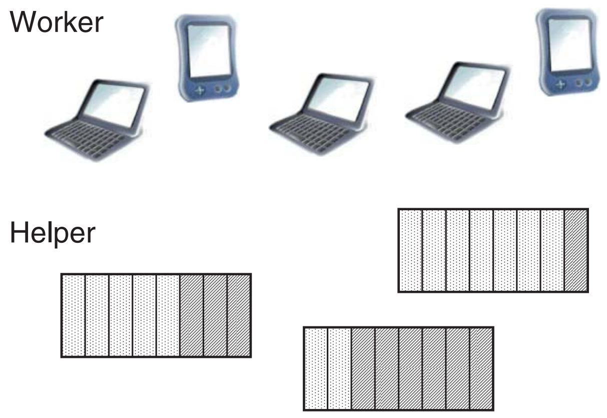


Fig. 1. Organic computing system with 5 worker components and 3 helper components, each helper has 8 slices each of which is configured for tasks of type 1 or type 2

is reconfigured so that it can perform all service tasks equally fast. It should be noted that we concentrate on scenarios where the reconfiguration time is large compared to the execution time of a service time. This is the most interesting case because when the reconfiguration times are relatively short it is possible to reconfigure the helper completely new for each task.

The paper is organized as follows. In the next Section II we introduce our model for a computing system. Details on the reconfiguration strategy and task distribution scheme are presented in Section III. An analysis of a two helper system is presented in IV. The experimental results are described in Section V. Conclusions are given in Section VI.

II. Model of the Computing System

In this section we formally describe our computing system model. A computing system consists of two types of components or members, ordinary members called *workers* and supporting members called *helpers*. Let m be the number of workers and $H_1, \ldots, H_n$ be the n helpers. Each helper has reconfigurable hardware (e.g., a Field Programmable Gate Array (FPGA)) on which it performs the service tasks. These hardware consists of q slices which can be reconfigured independently (see Figure 1 for an example). Each slice is always configured so that it works for only one type of tasks (it is assumed here that there exists two types of tasks). Different slices can be configured for different types of tasks. A reconfiguration (operation) has the effect that the type of task that can be executed on a slice is changed. During a reconfiguration a helper can reconfigure any number of $\leq q$ slices. The time of a complete reconfiguration of a helper is t_r and the time to reconfigure k slices is $(t_r/q) \cdot k$.

At each time step a helper can work for at most one task. The time it takes to finish a task depends on the number of slices that are reconfigured for the corresponding task type. A helper that has a proportion of $\geq 1/2$ of its slices configured for tasks of type i is called *specialized* for tasks of type i. It is *fully specialized* for tasks of type i when all slices are reconfigured for this type of tasks. Here we assume that the time is $t_e \cdot q/k$ where t_e is the execution time of a task on a fully specialized helper and k is the number of slices working for the task.

At each time step a worker needs servicing with probability $p_0 > 0$ before it can continue its normal work. The need for servicing can be of type 1 or type 2. A worker that needs servicing of type i searches for a helper and request a service task of type i. If the request is granted the service task is executed by the helper and afterwards the worker can continue its normal work. If the request is not granted the worker searches for another helper. It is assumed that searching here means that a random helper is contacted. The communication with the helper takes time $t_c \geq 0$ (no matter whether the request was successful or not).

A helper that gets a request for servicing always accepts the request when it has at least $q/2$ of its slices configured for the corresponding type of tasks. Otherwise, the probability that it accepts the request depends on a personal threshold value for this type of tasks, a stimulus value for the task, and the degree of specialization for this type of tasks. The *stimulus value* for a type of tasks is the number of tasks of this type minus the number of tasks of the other type counting all tasks that are actually requested by the workers. Let T_{ij}, $i \in \{1,2\}$, $j \in [1:n]$ denote the threshold of helper H_j for task type i and S_i the stimulus of task $i \in \{1,2\}$. The *degree of specialization* of a helper for a type of tasks is the relative number of slices that are configured for tasks of this type. Further let s_{ij}, $i \in \{1,2\}$, $j \in [1:n]$ denote the degree of specialization of helper H_j for task type i (when the context is clear indices i or j are omitted). The probability that helper H_j accepts a request for task i is defined as

$$p_{(s_{ij},S_i,T_{ij})} := \min\{1, f(s_{ij}) + \frac{S_i^2}{S_i^2 + T_{ij}^2}\} \qquad (1)$$

where the function $f(s_{ij})$ is defined in the next section.

III. Reconfiguration Strategy and Task Distribution

The reconfiguration strategy for the helpers that is investigated in this paper is motivated by stimulus-threshold models that include learning behaviour of insects. In the first strategy a helper that performs a servicing task always performs a reconfiguration operation so that the number of slices that can execute the corresponding type of servicing tasks is increased by one (unless all slices have already been configured for the corresponding type of tasks). We call this the *1-slice reconfiguration strategy*.

A possible problem of the 1-slice strategy is that the execution time of a servicing task is very long when only a few slices can execute it. Therefore we also studied a variant of the 1-slice strategy which is different for the case that a request for service is accepted when less than half of the slices are configured for the corresponding type of tasks. In this case the helper reconfigures itself so that half of the slices are configured for the accepted type before the execution starts. We call this the *1+half-slice reconfiguration strategy*.

Another important aspect is the strategy that is used by the helpers to decide whether a request for service should be accepted or not. As described in the last section a helper always accepts a request when it has at least half of the slices configured for the corresponding type of tasks. Otherwise it accepts with a probability that is determined by Formula 1. In the following we define the function f that is used in this formula.

For this we consider first the case of a single helper H and a static situation where it is assumed that the helper can not be reconfigured and the servicing requests arrive at constant rates for both types of tasks. The motivation for the definition of f is that H should reject so many tasks of the type is it not specialized for that it has an optimal reconfiguration for the resulting relative number of both task that it executes.

Therefore the first aim is to determine the optimal percentage of the slices of the helper that should be configured for type 1 tasks (the rest of the slices is then configured for tasks of type 2) for given fixed servicing probabilities for both type of tasks. Let $p > 0$ be the probability that a servicing task is of type 1. The run time of the tasks for a specialization level s for tasks of type 1 is $p/s + (1-p)/(1-s)$. It can be shown that the optimal percentage $g(p)$ of slices that should be configured for task 1 is $g(p) = (p - \sqrt{p-p^2})/(2p-1)$.

To define function f we consider a situation where it is assumed that the servicing requests of both types arrive at the same rate and where the stimulus value is $S = 0$. A helper H is considered which rejects tasks for which it is not specialized according to Formula 1. W.l.o.g. assume that these tasks are type 1. The function f is now determined so that the rate of tasks of type 1 that are accepted by H are optimal for the given configuration of H, i.e. f is defined such that for the relative number of accepted tasks of type 1 holds $g(1/(1+f(s)) = s$. It can be shown that $f(s) = (s-1)^2/s^2$.

More generally we can consider a situation where both tasks have different servicing probabilities. Assume that requests of type 1 (type 2) arrive with relative rate p (respectively $1-p$), $p \in [0,1]$. Then $f(p,s)$ is determined analogously as follows. Set $g(p/(p+(1-p)f(p,s)) = s$. Then $f(p,s) = \min\{1, (p/(1-p)) \cdot (s-1)^2/s^2\}$.

Clearly, our definition of f is heuristic and not necessarily optimal for a computing system with several helpers.

IV. Analysis of a Two Helper Computing System

In this section we analyze the computing system with two helpers analytically. For the analysis a static situation is assumed where the request rates for servicing tasks do not change and where helpers are not reconfigured. It is assumed that exactly half of the requests for each type of tasks are for each of the two helpers H_1 and H_2. Let p be the proportion of requests for servicing of type 1. Thus the arrival rate of requests for tasks of type 1 (type 2) to H_1 is $1/2 \cdot p$ (respectively $1/2 \cdot (1-p)$). The execution time of each servicing task on a fully specialized helper is assumed to be 1. If a request for a task is rejected then it is always send to the other helper (this slight deviation from the model described in Section II is assumed to make the analysis easier). The aim is to find out how much of the arriving requests should be accepted in order to minimize the total execution and communication time. W.l.o.g. we can assume that each type of request can be rejected by at most one of the helpers. Assume that request for tasks of type i are always accepted by helper H_i, $i \in [1,2]$. Let s_i be the proportion of slices configured for tasks of type i of helper H_i. Let $f_i : [0,1] \to [0,1]$ be the function that determines the fraction of accepted requests on helper H_i for tasks of type j, $j \neq i$, $i,j \in \{1,2\}$, depending on the specialization level s_i of H_i.

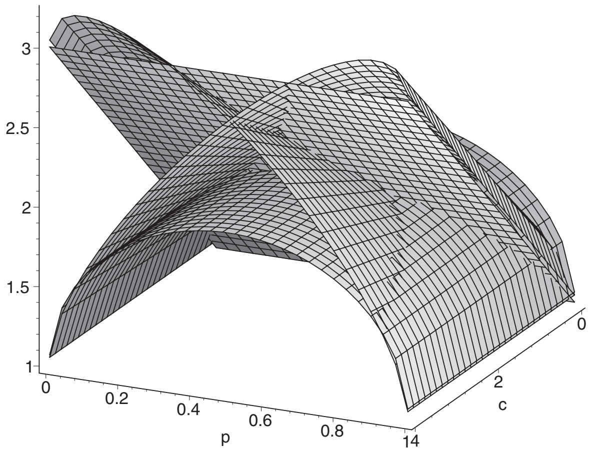


Fig. 2. Two helper computing system: hyperplanes showing the total execution and the additional communication costs (i.e., the communication costs minus c — the reason that we do not add all communication costs is to make the figure better readable) corresponding to cases i-iv of Section IV

The total execution time e_1 for all tasks that are executed on helper H_1 is

$$e_1 := \frac{1}{2} \cdot \left(\frac{p + p(1 - f_2(s_2))}{s_1} + \frac{(1-p)f_1(s_1)}{1-s_1} \right)$$

This time consist of the execution time $(1/2) \cdot (p/s_1)$ for the requests of type 1 that arrive directly at H_1, the execution time $(1/2) \cdot p(1-f_2(s_2))/s_1$ for the requests of type 1 that have been rejected by H_2, and the execution time $(1-p)f_1(s_1)/(1-s_1)$ for the requests of type 2 that arrive directly at H_1 and are accepted.

Similarly the total execution time e_2 for all tasks that are executed on helper H_2 is

$$e_2 := \frac{1}{2} \cdot \left(\frac{pf_2(s_2)}{1-s_2} + \frac{(1-p) + (1-p)(1-f_1(s_1))}{s_2} \right)$$

The communication costs ω are

$$\omega := c + \frac{1}{2} \cdot c \cdot (p(1 - f_2(s_2)) + (1-p)(1 - f_1(s_1))).$$

The communication cost consists of the costs for the first communication for each request plus the costs for the second communication that is necessary when a request was rejected. The total execution and communication times C for all tasks is $C = e_1 + e_2 + \omega$.

In order to find an optimal (with respect to minimal total execution and communication costs) degree of specialization

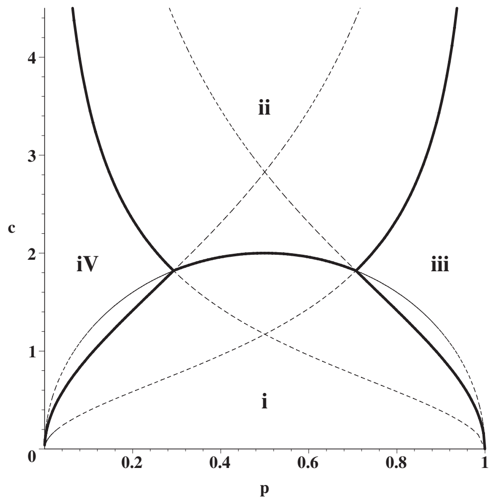


Fig. 3. Two helper computing system: cutlines between pairs of hyperplanes for total execution and communication costs for cases i-iv; numbers indicate which of the cases i-iv is optimal in the corresponding region

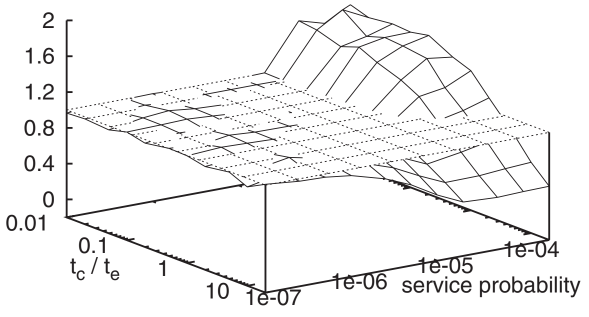


Fig. 4. Relative throughput of the self-organized computing system compared to a $s = 0.5$-$f = 1$-system

for both helpers it can easily be seen that it is optimal for the computing system when helper H_i either rejects all request for tasks of type j, $j \neq i$ or it accepts all these tasks. Therefore it holds $f_i(s_i) = 0$ or $f_i(s_i) = 1$ for $i \in \{1, 2\}$ and all $s_i \in [0, 1]$. The following four cases can occur:

i. $f_1 = 0$, $f_2 = 0$:
In that case each helper H_i rejects all requests for tasks of type j, $j \neq i$. To achieve a minimal total execution and communication time it is best when both helpers are maximally specialized, i.e., $s_i = 1$, $i \in \{1, 2\}$. Then $C := 1 + 3/2 \cdot c$.

ii. $f_1 = 1$, $f_2 = 1$:
Both helpers accept all requests and no additional communication occurs. It is easy to see that for the optimal specialization level $s_1 = 1 - s_2$ holds. Further, for arrival rates p and $(1 - p)$ follows $s_1 = 1 - s_2 = g(p) = (p - \sqrt{p - p^2})/(2p - 1)$ as shown in Section III. Then $C := (2p - 1)^2 \sqrt{p(1 - p)}/(p - \sqrt{p(1 - p)})(p - 1 + \sqrt{p(1 - p)}) + c$. Note, that $C = 2 + c$ for $p = 1/2$ and $\lim_{p=1} C = 1 + c$.

iii. $f_1 = 1$, $f_2 = 0$:
In this case H_2 rejects all request for tasks of type 1. Therefore it is optimal when it is fully specialized for task 2 ($s_2 = 1$). For H_1 this leads to arrival rates of $1/2 \cdot p + 1/2 \cdot p$ for requests of type 1 and $1/2 \cdot (1 - p)$ for requests of type 2. Hence the relative rate for requests of type 1 (type 2) is $2 \cdot p/(p + 1)$ (respectively $(1 - p)/(p + 1)$). Then the optimal value for s_1 is $s_1 = g(2p/(p + 1))$. The formula for the resulting total execution an communication costs is omitted because it is lengthy (the cost values are shown in Figure 2).

iv. $f_1 = 0$ and $f_2 = 1$:
Analogous to case iii.

Figure 2 shows the total execution and communication costs for all four cases. It can be seen from Figure 3 that there exists for each of the cases i-iv a region of values for parameters c and p where the case is optimal. For large communication costs and values of p that are neither very small nor very large it is optimal when all requests are accepted. If the communication costs are small and the values of p are not too extreme then it is optimal when each helper rejects one type of requests. For large (small) values of p it is optimal when H_1 rejects (respectively accepts) all requests of type 2 and H_2 accepts (respectively rejects) all requests of type 1.

It should be noted that for practical purposes it could be important to consider the relative amount of work that is done by both helpers. In order to make the amount of work done by both helpers similar it can be necessary to use f functions that are not only always zero or one.

V. Experiments

If not stated otherwise we use in all our experiments 2 types of tasks and the execution time of a task is $t_e = 100$ for a helper task that is fully specialized. The communication cost were set to $t_c = 10$. The number of workers was set to $m = 100$ and the number of helpers is $n = 10$. The helpers have 10 reconfigurable slices and the time to reconfigure the helper completely is $t_r = 1000 (t_r/t_e = 10)$. The thresholds of all helpers is $T = 100$. All results are averaged over 20 runs.

In this section we denote by $s = x - f = y$-system with $x \in [0, 1]$, $y \in [0, 1]$ a simple system where the degree of specialization is fixed to x for each helper and the value of the f function for x is equal to y. Note, that the system with fixed parameter s performs no reconfiguration operations and therefore has no reconfiguration costs.

A. Static Environment

The communication costs t_c are an important parameter for the behaviour of the system. Their influence for the case that both types of requests have identical probabilities is investigated in the following.

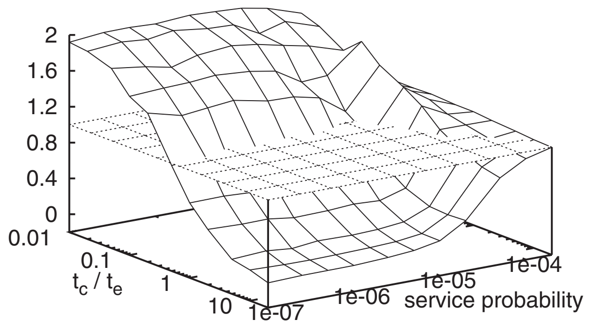


Fig. 5. Relative sojourn times of the self-organized computing system compared to a $s = 0.5$-$f = 1$-system

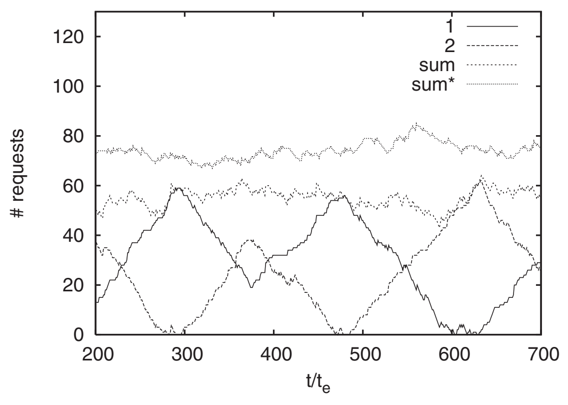


Fig. 6. Number of actual request in a one helper system compared to a one helper $s = 0.5$-$f = 1.0$-system; i: number of request of type i, sum (sum*): total actual number of requests (for the $s = 0.5$-$f = 1.0$-system)

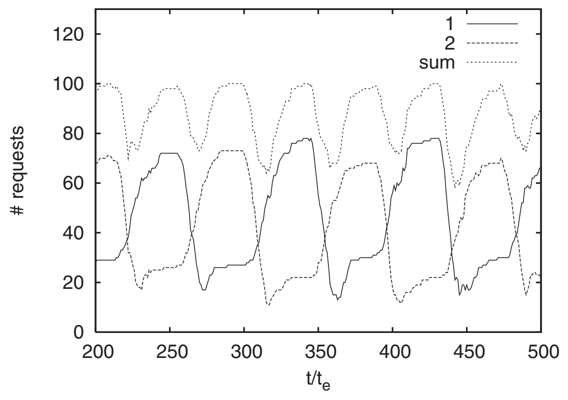


Fig. 7. Number of actual request in a ten helper system; i: number of request of type i, sum: total actual number of requests; probability of service is 0.001 per time step and type of service, $t_e = 100$, $t_r/t_e = 30$

Figure 4 compares the throughput (i.e., the number of servicing tasks that have been finished in a certain time) of the system with self-organized task allocation with a $s = 0.5$-$f = 1$-system. Shown is the relative performance difference of both systems for service probabilities in $\{0.001,\ 0.0005, 0.0002, \ldots, 0.000001\}$ and relative communication costs $t_c/t_e \in \{0.01,\ 0.02, \ldots, 20\}$. For high service probabilities and small values of t_c/t_e (< 1), the throughput of the self-organized computing system is almost twice as high as for the $s = 0.5$-$f = 1$-system. Only for high service probabilities and very high relative communication costs ($t_c/t_e \gtrsim 2$) the throughput of the $s = 0.5$-$f = 1$-system is better. The reason is that in the self-organized system requests are rejected with some probabilities, which implies additional communication costs. For small service probabilities and relative communication costs of $t_c/t_e \lesssim 10$ the performance of both systems is similar.

Figure 5 compares the sojourn times (i.e., for each tasks the time from its first request to the end of its execution time is measured) of the servicing requests in the self-organized system and the $s = 0.5$-$f = 1$-system. The performance of both systems differ significantly for most parameter combinations. For small values of t_c/t_e (< 1) the sojourn times of the self-organized computing system are smaller for all tested probabilities of service. Only for high service probabilities and very high relative communication costs ($t_c/t_e \gtrsim 2$) the sojourn of the $s = 0.5$-$f = 1$-system are better.

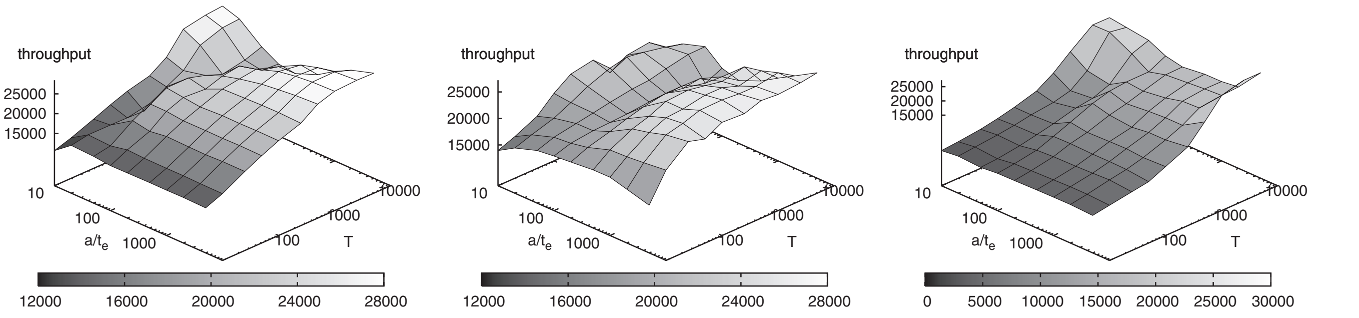
In the following we demonstrate an oscillation effect that is typically for many self-organized systems. This effect occurs in a simple system with only one helper that has one slice. The probability that a worker needs service is 0.0001 per time for each type of service. The reconfiguration costs are $t_r = 1000$ and the threshold parameter was set to $T = 1000$.

Figure 6 shows the number k_1 (k_2) of actual requests for tasks of type 1 (respectively type 2) over time. It can be seen, that the values of k_1 and k_2 are oscillating. The reason for this is that the helper specializes for one type of tasks — say type 1 — and then rejects requests for the other type (with a probability defined by the f function). When the number of (waiting) requests for type 2 increases, the stimulus increases also. This leads to a increased probability that the helper executes tasks of type 2. The reconfiguration operations that are done in this case have the effect that the helper becomes specialized for tasks of type 2. Compared to a simple $s = 0.5$-$f = 1.0$-system with one helper the figure shows that the actual number $k_1 + k_2$ of requests that are waiting is smaller.

The oscillating behavior occurs also in a system with more than one helper. In the case of high reconfiguration costs the system reacts slowly. This can be seen in Figure 7 for a system with $n = 10$ helpers and where the reconfiguration time is 30 times larger than t_e.

B. Environment with Changing Service Probabilities

To investigate the behavior of the self-organized task allocation scheme in more dynamic situations we used changing servicing probabilities. In the experiments described in the following the service probabilities 0.0004 and 0.0016 were exchanged every $a/t_e = \{10, 20, 50, 100, 200, 500, 1000, 2000, 5000\}$ time



(a) 1 slice (b) 10 slices (c) 100 slices

Fig. 8. Self-organized computing system: throughput for different thresholds T and different degrees of dynamics a; $t_e = 100$, $t_r = 1000$ and $t_c = 10$

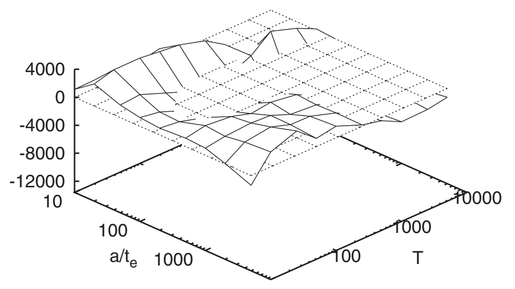
steps. Obviously the threshold parameter T has a strong influence on the adaptiveness of the computing systems. For a high value for T it is unlikely that the helpers reconfigure themselves (comp. Equation 1). In the experiments threshold values $T \in \{10, 20, 50, 100, 200, 500, 1000, 2000, 10000\}$ were used. For each combination of a/t_e and T the throughput within $3000 * t_e$ time steps was measured. Note, that $a/t_e = 2000$ is a situation where the servicing probability is changed only once over the considered time interval. The reconfiguration time was set to $t_r = 1000$ and communication time $t_c = 10$ was used in the experiments. All experiments have also been done with $t_c = 100$. But the results are not presented because they are very similar to those for the smaller communication times.

The experimental results for the self-organized service system are depicted in Figure 8 for reconfiguration time $t_r/t_e = 10$, $t_c/t_e = 0.1$ and different number of slices $q \in \{1, 10, 100\}$. The throughput of the self-organized system has to be compared to the average throughput of 14109 achieved with the $s = 0.5$-$f = 1.0$-system.

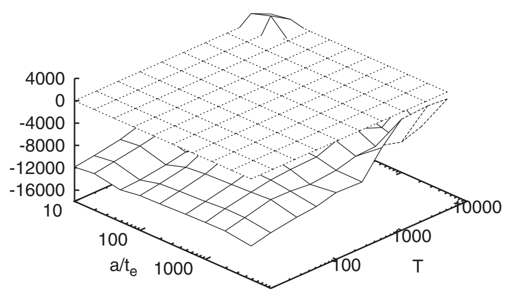
For very large values of T the self-organized service system is not very adaptive. In all tested cases the best performance is achieved for the extreme values of $a/t_e = 10$ or $a/t_e = 2000$. The reason is that using $a/t_e = 2000$ does not require adaptiveness. When $a/t_e = 10$ is used the arrival rates change so often, that the situation becomes similar to a situation where the probabilities for both type of service are identical and fixed. Such a situation also does not require adaptiveness and hence a high value of T leads to a good performance.

The worst throughput is achieved in situations with many slices and small value for T. In such situations it is likely that a helper excepts a request, even when it has a small degree of specialization for the corresponding type of tasks. This can lead to very large execution times of the tasks (recall that using only 1 out of k slices for a task increases the execution time by factor k compared to an execution with full specialization).

Figure 9 compares the results of the standard self-organized system (using the 1-slice reconfiguration strategy) with one where the 1+half-slice reconfiguration strategy is used. Since both strategies are the same for 1 slice results are shown only for systems with 10 and 100 slices. The motivation to introduce the 1+half-slice reconfiguration strategy was to make the system faster adaptive a changes of the relative servicing probabilities for different types of tasks. It can be seen in the figure that the 1+half-slice reconfiguration strategy obtains for 10 slices a higher throughput higher threshold values (and not too small values of a, recall that $a/t_e \leq 30$ leads to a situation that is similar the a situation with constant service probabilities). For 100 slices (where a higher adaptivity is even more necessary) the 1+half-slice reconfiguration strategy

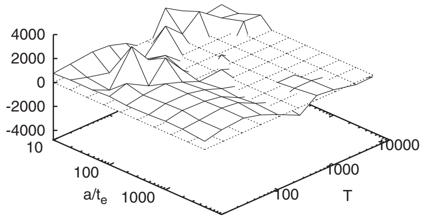


(a) 10 slices

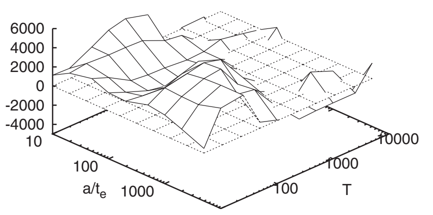


(b) 100 slices

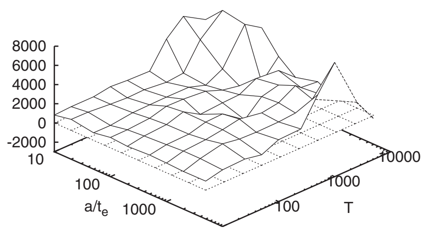
Fig. 9. Self-organized service system: throughput difference between the standard system (using the 1-slice reconfiguration strategy) and a system using the 1+half-slice reconfiguration strategy for different thresholds T and different degrees of dynamics a; $t_e = 100$, $t_r = 1000$ and $t_c = 10$



(a) 1 slices



(b) 10 slices



(c) 100 slices

Fig. 10. Self-organized service system: throughput difference between the standard system (with the same threshold value T for each helper) and a system using diverse-threshold method for different thresholds T and different degrees of dynamics a; $t_e = 100$, $t_r = 1000$ and $t_c = 10$

is better than the standard reconfiguration strategy. Only for situation where the threshold values are very high ($T > 5000$) and very small values of $a/t_e \leq 30$ the standard reconfiguration strategy is better.

A possible problem in a dynamic situation when all helper use the same threshold value is that they might start all at the same time to change their specialization. Instead it might be better when the helper change their specialization one after the other as long as it is necessary for the new situation. To test this we used a variant of the self-organized computing system where each helper has a different threshold value. We call this the *diverse-threshold method*. For the diverse-threshold method we used fixed thresholds of $(j \cdot T)/(n+1)$ for helper H_j. Figure 10 compares the results of the standard self-organized system to the variant with the diverse-threshold method. The results show that the system using the diverse-threshold method obtains a better throughput only for 1 or 10 slices and compared to a standard system with high threshold values. For 100 slices the the standard system works better.

VI. Conclusions

We have proposed a self-organized scheme of the allocation of service tasks for an adaptive computing system. In our model the computing system consists of normal worker components and helper components where the workers need some service from time to time in order continue with their normal work. The service is done by the helpers which have reconfigurable hardware to perform the different service tasks. The speed of service for a certain task depends on the amount of resources configured for this task by the helper. We have studied different strategies that can be used by the helpers to decide whether they should accept a service task and whether they should reconfigure themselves. These strategies are inspired by stimulus-threshold systems that have used in the literature to explain task allocation in social insects. For a simple two helper system we have presented analytical results. For systems with a larger number of helpers we have presented experimental results. They show for example that these systems can adapt to dynamic situations with changing probabilities for service.

Future work is to investigate more elaborated strategies where the helpers collect information about their environment and make reconfiguration decisions during the runtime of a tasks. Also systems where the helpers are partially reconfigurable and where they can execute several tasks at the same time will be studied.

Acknowledgement

This work was supported by the German Research Foundation (DFG) through the project “Organisation and Control of Self-Organising Systems in Technical Compounds” within SPP 1183.

References

[1] W. Agassounon, A. Martinoli: Efficiency and Robustness of Threshold-Based Distributed Allocation Algorithms in Multi-Agent Systems. In Proc. of the First Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems AAMAS-02, ACM Press, 1090-1097, 2002.

[2] E. Bonabeau, G. Theraulaz, and J.-L. Deneubourg: Fixed response thresholds and the regulation of division of labor in insect societies. Bulletin of Mathematical Biology, 60:753–807, 1998.

[3] E. Bonabeau, A. Sobkowski, G. Theraulaz, J.-L. Deneubourg: Adaptive Task Allocation Inspired by a Model of Division of Labor in Social Insects. In D Lundh et al. (Eds,), Biocomputing and Emergent Computation, World Scientific, 36–45, 1997.

[4] E. Bonabeau, G. Theraulaz, and J.-L. Deneubourg: Quantitative study of the fixed threshold model for the regulation of division of labour in insect societies. Proc. Roy. Soc. London B 263, 1565-1569. 1996.

[5] V. A. Cicirello, S. F. Smith: Wasp-like Agents for Distributed Factory Coordination. Autonomous Agents and Multi-Agent Systems 8(3): 237-266, 2004.

[6] V. A. Cicirello, S. F. Smith: Distributed Coordination of Resources via Wasp-Like Agents. Proc. First NASA GSFC/JPL Workshop on Radical Agent Concepts (WRAC), 71-80, 2003.

[7] V. A. Cicirello, S. F. Smith: Wasp nests for self-configurable factories. Proc. Fifth Int. Conf. on Autonomous Agents , 473-480, 2001.

[8] V. A. Cicirello, S. F. Smith: Insect Societies and Manufacturing IJCAI-01 Workshop on Artificial Intelligence and Manufacturing, 2001.

[9] P.R. Ferreira, D. de Oliveira, and A.L.C. Bazzan: A Swarmbased Approach to Adapt the Structural Dimension of Agents’ Organization. Journal of the Brazilian Computer Society, 11(1): 63-73, 2005.

[10] J. Gautrais, G. Theraulaz, J.-L. Deneubourg, and C. Anderson: Emergent polyethism as a consequence of increased colony size in insect societies. Journal of Theoretical Biology, 215: 363-373, 2002.

[11] GI: Organic Computing / VDE, ITG, GI - Positionspapier. 2003, online: http://www.betriebssysteme.org/Betriebssysteme/FutureTrends/oc-positionspapier.pdf

[12] J.O. Kephart, D.M. Chess: The Vision of Autonomic Computing. IEEE Computer, 36(1): 41-50, 2003.

[13] O. Kittithreerapronchai, C. Anderson: Do ants paint trucks better than chickens? Market versus response thresholds for distributed dynamic scheduling. Proc. IEEE Congress on Evolutionary Computation, 2003.
[14] M.J.B. Krieger, J.-B. Billeter: The call of duty: Selforganised task allocation in a population of up to twelve mobile robots. Robotics Autonom. Sys. 30: 65-84, 2000.
[15] D. Merkle, M. Middendorf: Dynamic Polyethism and Competition for Tasks in Threshold Reinforcement Models of Social Insects. Adaptive Behavior, 12: 251-262, 2004.
[16] R. Price, P. Tino: Evaluation of Adaptive Nature Inspired Task Allocation Against Alternate Decentralised Multiagent Strategies. Proc. Parallel Problem Solving from Nature - PPSN VIII, 982-990, LNCS 3242, Springer, 2004.
[17] G. Theraulaz, E. Bonabeau, and J. Deneubourg: Response threshold reinforcement and division of labour in insect societies, Proc. Roy. Soc. London B 265, 327-332, 1998.