Autonomous Distributed System Based on Behavioral Model of Social Insects

Daichi Teruya[1], Bipin Indurkhya[2], Tadakatsu Maksaki[3] and Hironori Nakajo[4]
[1]Department of Computer and Information Sciences Tokyo University of Agriculture and Technology
Koganei-shi, Tokyo, 184-8588 Japan
[2]Department of Computer Science AGH University of Science and Technology
Al. Mickiewicza 30, 30-059 Kraków, Poland
[3]Department of Media Information Engineering National Institute of Technology, Okinawa-collage
905, Henoko, Nago-shi, Okinawa, 905-2171, Japan
[4]Institute of Engineering Tokyo University of Agriculture and Technology
Koganei-shi, Tokyo, 184-8588 Japan

Abstract— *Social insects are self-organized living organism without a commanding system with a single leader. A model which realizes self-organization of social insects can be constructed using a reaction threshold model based on the concept of Stigmergy. This paper proposes such a model of new autonomous distributed system using a behavioral model of social insects. This model is able to allocate autonomous computation resources and retain fault tolerance without the control of a commanding manager. Our evaluation results show that the proposed model works as an autonomous distributed system and demonstrates its effectiveness for fault tolerance. A problem with respect to unequal resource distribution was found in the experimental model, so we proposed an improved method.*

Keywords: Autonomous distributed system, Fault tolerance, Reaction threshold model

1. Introduction

1.1 The ecology of social insects and its applications

Social insects such as ants and bees construct colonies composed of a very large number of individuals. These colonies are not dominated by a commanding manager to distribute labor for the necessary colony-management tasks among the members. However, considering the overall organization of the colony, necessary labor tasks are divided among the colony members adaptively as shown in Fig.1[1] [2].

Ishii et al. have made two observations in a colony of *Myrmica kotokui*, where the individual workloads vary quite a bit [3]: "When only individuals which do not work are collected, some working individuals will appear," and "when only individuals which work hard are collected, some loafing individuals will appear." These observations show the existence of an adaptive control system of labor, in which some individual workers take rest depending on the situation. This control system is modeled mathematically by Bonabeau et al. as “Response Threshold Model” [2]. The intensity of a stimulus to which an ant starts to react is called "Reaction Threshold", which varies greatly among individuals. Because of this property, adaptive task distribution can be realized among individuals.

Taking larva feeding as an example, each adult has a threshold n “to start feeding larvae when n larvae covet food.” When there are only a few larvae coveting food, only low-threshold adults work to feed them. When the number of coveting larvae increases, adults with a higher threshold also join in together with the low-threshold adults.

Self-Organization of social insects is realized by information transmission via environment, which is called *Stigmergy,*. M. Dorigo et al. proposes an optimization method as “Ant Colony System” for traveling salesman problem[4]. When an ant brings food to its own nest, it leaves a volatile pheromone on the route. The following ants can find food by tracing the pheromone.

When ants use multiple routes to collect foods, the pheromone on a shorter route has a higher density than the pheromone on a longer route, which results in more ants selecting the shorter route. Over time, as fewer and fewer ants select the longer paths, their pheromone density decreases, and eventually the path is eliminated. This algorithm is widely applied in information systems as “Ant Colony Optimization” [5] [6].

1.2 Applications in distributed system

As computer systems are becoming more and more complex, setting and managing network configuration has become a major problem. Therefore, the concept of “Autonomic Computing”, according to which a computer-system configuration optimizes itself like a human body, has been proposed [7]. In this approach, we focus on the commonality between the task-allocation mechanism of social insects and

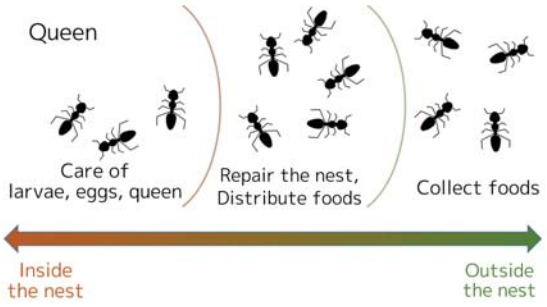

Fig. 1: Organization structure of ants

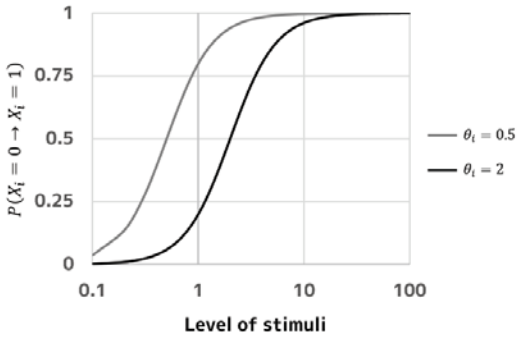

Fig. 2: Comparison of response curves with different θ_i

the resource-allocation problem in a distributed system to develop autonomic-computing distributed systems based on social-insect colonies.

Coordinating robots is one successful example that applies social insect ecology in the field of autonomous distributed system. In the TERMES project, multiple robots, which have only their own sensor information, collaborate to build a common construction [8] [9]. Another example is provided by Byrski et al. [10], where socio-cognitive features of a population related to perspective taking are incorporated into Ant Colony Optimization to improve performance.

The Ministry of Economy, Trade and Industry of Japan predicts that computer systems are going to be shifted to a decentralized system with expanding IoT/CPS in the future[11]. In their report, an innovative distributed system is expected to maintain zero downtime, and to reduce load in increasing communication bandwidth with collaboration among IoT devices, distributed clouds and fog by eliminating the core node that monitors and controls all nodes.

1.3 Goals

We aim to tackle the problem of dynamically allocating tasks to multiple computing resources by designing an architecture based on Stigmergy and the Response Threshold model. In particular, we would like our architecture to meet the following three goals:

- *Distributed computing without a central node*
- *Dynamic allocation of computing resources according to task status*
- *Robustness against failures in computing nodes*

In this paper, we describe such an architecture for a distributed system and evaluate it.

In the following section, we describe in detail the biological model used in this paper. In Section 3, we model an autonomous distributed system using this biological model. Verification of the effects of the proposed model is presented in Section 4. Finally, we present our conclusions and evaluation of the proposed model in Section 5.

In the following section, we describe in detail the biological models used in this paper. In section 3, we model an autonomous distributed system using these biological models. In section 4, we discuss related works and the distinguishing characteristics of our research. Verification of the effects of the proposed model is presented in Section 5. Finally, we present our conclusions and evaluation of the proposed model in Section 5.

2. Applying biological models

2.1 Response Threshold model

Tasks in a society of ants can be categorized into two types: constantly occurring tasks such as caring for larvae, and suddenly occurring unpredictable tasks such as food collection.

Since the number of tasks and the amount of labor required for each task at a certain moment are difficult to predict, it is necessary to allocate an appropriate number of workers to each task existing at the time. As ants form huge colonies, a statically organized regime with a single commanding manager that requires command transmission is not optimal. Bonabeau et al. found that each ant responds to a task when the intensity of the stimulus brought by the task exceeds the threshold value for that ant[12].

Let X be the state of an individual ($X = 0$ corresponds to inactivity and $X = 1$ corresponds to full activity for the task), and let θ_i be the Reaction Threshold of individual i, when the degree of stimulus s is given to the individual i, the probability P to execute the task is defined as follows[2].

$$P(X_i = 0 \to X_i = 1) = \frac{s^2}{s^2 + \theta_i^2} \quad (1)$$

Fig.2 shows a graph of changing reaction probability P when the Reaction Threshold θ_i is small and large. From the graph, we can see that as θ_i becomes larger, the probability P changes against the strength of the stimulus more slowly.

In this paper, according to Bonabeau's model, we define the probability function $P_{ij}(t)$, which represents the probability of the case where the worker i reacts to the task j at the time t as follows:

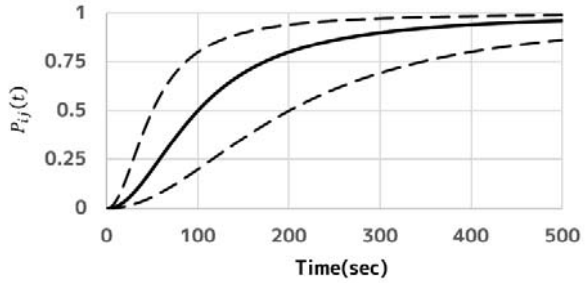

Fig. 3: Graph of $P_{ij}(t)$ using Equation(3) ($\bar{T} = 100$)(upper:$\theta_i = 0.5$, center:$\theta_i = 1$, lower:$\theta_i = 2.0$)

$$P_{ij}(t) = \frac{(\theta_i \cdot S(M_i, t))^2}{(\theta_i \cdot S(M_i, t))^2 + \bar{T}^2} \tag{2}$$

Here θ_i represents the Reaction Threshold unique to the worker i. It expresses the ease of reaction to the task. The default value of θ_i is set to 1. A smaller value of θ_i means that it is harder for the worker to start the task, and a larger value of θ_i means that it is easier to start the work. $S(M_j, t)$ is the urgency of request to process of task j at time t. The larger the value, the higher is the need for processing. M_j is additional information unique to the task j. It is necessary for the worker to calculate the function $S(M_j, t)$. $\bar{T}$ is a constant to adjust the changing speed of P_{ij} throughout the system; it represents the time until the reaction probability of a worker with $\theta_i = 1$ becomes 0.5.

· An example of function S

Function $S(M_j, t)$ can be defined by arbitrary expressions according to the change in the request patterns. For example, if the requests increase linearly with time, we can define the function $S(M_j, t)$ as follows, with the additional information M_j being the time T_j when the task is given:

$$S(T_j, t) = A \cdot (t - T_j) \tag{3}$$

A is a constant introduced to adjust the rate of change in the requests. In addition, by defining $S(M_j, t)$ as required, to include tasks that have some requests regardless of time and tasks that wait for some time for requests, the expected task distribution can be realized.

2.2 Concept of Stigmergy

Direct communication like P2P among individuals becomes inefficient when the colony becomes large; it also becomes difficult to realize self-organization of the colony. Therefore, social insects use an environment-based information transmission mechanism called "Stigmergy" as the basis for worker-allocation systems [1]. In the model proposed in this paper, we introduce this information transfer mechanism to build an infrastructure for self-organization. A subsystem for information storage, *Stigmergy,* is allocated, which is equivalent to "environment". In the rest of this paper, "Stigmergy" refers to this subsystem in the proposed model.

3. Developing a model of the new autonomous distributed system

In this section, we propose a new autonomous distributed system using the Response Threshold model based on the concept of Stigmergy. Components of this system and its flow of operation are shown in Fig.4.

3.1 Job and Task

The unit of request given to the system by a user is called a **Job**, which is divided into **Tasks** representing small problems to which a unique ID j and additional information M_j are added.

3.2 Components of the model

The system in the proposed model consists of three components: **Worker**, **Stigmergy**, and **Mediator**. The components can be implemented in a single computer system, or can be spread across multiple computers.

3.2.1 Worker

Multiple Workers can be implemented to process tasks in a system. Each Worker has a parameter θ_i representing its Reaction Threshold, as well as a random number generator. The value of θ_i is set in a Worker using uniform random numbers or normal random numbers. However, range of values, average, variance etc. are unified across all the Workers.

3.2.2 Stigmergy

Stigmergy is implemented uniquely in each system, and it represents the *environment* for sharing information. Stigmergy holds 'waiting-task information' and 'processed-task information.' The processed-task information is composed of a unique task ID j, and the result of the task. The Stigmergy subsystem supports the following operations:

- Obtaining a list of waiting tasks
- Obtaining a list of processed tasks
- Obtaining and recording the task information and the task results

3.2.3 Mediator

There can be multiple Mediators in a system. A Mediator is an interface between the system and its user. A user can operate the Mediator as necessary to post a Job and browse its processing results. In addition, the Mediators handles partitioning a Job into tasks and collecting the processed results from the tasks.

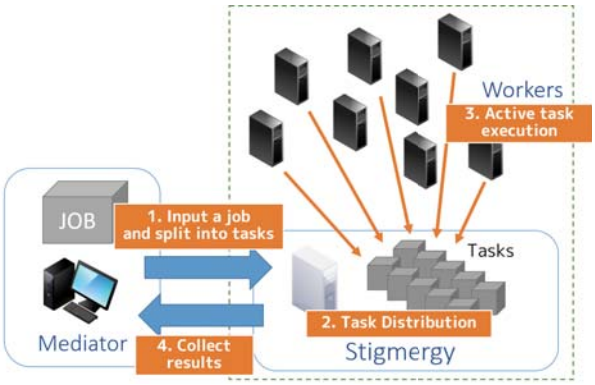

Fig. 4: Flow of Job processing

3.3 Flow of Job processing

Step 1. Input Jobs and divide into tasks

After Mediator divides a Job given by the user into tasks, a task-specific identifier j and additional information M_j are added to each spawned task. Spawned tasks are transferred to Stigmergy.

Step 2. Task distribution via Stigmergy

The Worker regularly inquires Stigmergy for the list of waiting tasks. At this time, the Worker calculates the response probability $P_{ij}(t)$ for all tasks. If the uniform random number value r, which varies from 0 to 1, is $r < P_{ij}(t)$, the Worker acquires the task j from Stigmergy for processing.

Step 3. Task processing by Worker

The Worker, on receiving a task from Stigmergy, begins to process the task. When the task is completed, the Worker transfers the result to Stigmergy. When Stigmergy saves the results of the processed task, it deletes the task from the list of waiting tasks and adds it to the list of processed tasks.

Step 4. Collecting processed results

The Mediator periodically queries Stigmergy for the processed task list. The Mediator requests the results from task processing as necessary, and collects them. When all the results are collected, they are outputted.

4. Related Works

This section discusses related works and the distinguishing characteristics of our research with respect to these works.

4.1 Existing autonomous decentralized systems

Many autonomous distributed computing systems have been researched so far. A data-sharing mechanism like Stigmergy to realize a distributed system has already been proposed[13]. Another piece of research that takes an approach similar to ours is the Data Field Architecture[14][15] proposed by Hitachi, Ltd. This system has subsystems called "Data Field" corresponding to Stigmergy and "Atom" corresponding to a Worker. Atom handles the data shared at Data Field. The Data Field Architecture is aimed to realize the following goals:

- Fault-Tolerance
- On-line Expansion
- On-line Maintenance

Our research is based on applying the Reactive Threshold model to a system architecture like Data Field Architecture to exhibit the following characteristics:

- Auto Scaling
- Adaptive Task Allocation
- Setting Priority for each Task/Job

4.1.1 Auto Scaling

When using the Reaction Threshold model, both types, the "diligent Worker" and the "lazy Worker", emerge because the value of θ is different for each Worker. When the number of tasks per Worker begins to increase, the request size of the task tends to increase, and some of the "lazy Workers" start to work. In this way, redundant Workers are automatically allocated to continue task processing with the minimum number of required Workers, by auto scaling according to the number of tasks.

4.1.2 Adaptive Task Allocation

Our proposed method is a dynamic load-balancing system. Eager et al. show that dynamic load balancing performs better than static load balancing[16]. In our system, it is possible to realize dynamic load balancing considering the processing capability of each Worker without central nodes, while acquiring the features described in this section.

4.1.3 Setting Priority for each Task/Job

A processing priority is assigned to each task or Job. For example, if it is assumed that the value of A in Expression(3) is given individually to each Job, the Request Value S of a task with a larger value of A tends to become larger, thereby increasing the priority of that task.

When the Request Value is set to be sufficiently large, regardless of the elapsed time, no ''lazy Worker" will emerge and all Workers will work actively. In such a situation, the system exhibits the same characteristics as in the previous research [14][15].

4.2 Using stochastic model

"Load Sharing Algorithm" is a conventional method of managing a distributed system using probability. It uses dynamic load sharing which can be applied even in heterogeneous systems.

Table 1: Computers used for the experiment

	A	B
Machine	Raspberry Pi 2 Model B	Raspberry Pi 3 Model B
CPU	ARM Cortex-A7 4cores@900MHz	ARM Cortex-A53 4cores@1.2GHz
Memory	1GB	1GB
OS	Raspbian (Mar. 2017)	Raspbian (Mar. 2017)

However, in a Load Sharing Algorithm, since each node needs to negotiate with other nodes, the task distribution requires some iterations to be stabilized. On the contrary, our system decides whether each node executes a task independently based on a simple sigmoid function. Therefore, negotiation with other nodes is not required. Furthermore, since an unprocessed Job is not transferred, a smaller network bandwidth is required.

Our model is easy to analyze because it is equivalent to an $\mathrm{M/M/K}$ typed queue, so it is advantageous to apply it to a practical system.

5. Verification of the model

This section explains validation of the proposed model for verification with experiments especially on each of the three new advantages: auto scaling, adaptive task allocation, and setting priority for each task/job as mentioned in Section 4.1.

5.1 Environment of implementation and experiment

The experimental system is implemented using C++11. The communication between the components is realized by TCP/IP using sys/socket.h. There are two types of computers, called A and B, used in the experiment, as shown in Table 1.

We utilize computer A as Stigmergy and Mediator, and 10 nodes of computer B as Workers. These are connected using a router and a switching hub as shown in Fig.5.

5.2 Experiment model

5.2.1 System structure

A single Mediator and multiple Workers are implemented to collaborate via Stigmergy in the experimental system. It is assumed that four Workers can operate as different processes in each computer.

5.2.2 Job and Task Settings

Brute force attacks against RC4 cryptography is adopted as an example to evaluate our system. In the implementation, a Job is given with the following contents:

- Length of a key (Byte) - N_k
- Number of divisions - N_d

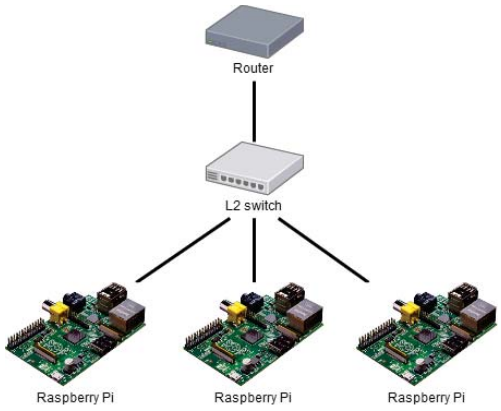

Fig. 5: Network structure for the experiment

- Plain text
- Cipher text

Since the search range of the key is divided into N_d pieces and allocated to each Task, the number of keys to be searched for one task is $N_r = N_k/N_d$. A serial number is given to each Job as its ID. The time T_j at the time when the task j is divided is given to the task j as the additional information M_j. The Mediator generates a task ID combining the ID of the source Job and a serial number. A task is composed of the following contents:

- Length of a key (Byte)
- Length of the search range - N_r
- Plain text
- Cipher text

The function S, which expresses the degree of request of Task j, is set to $S(A_j, T_j, t) = A_j \cdot (t - T_j)$. A_j is the priority of task j and T_j is the time at which Task j is generated.

5.2.3 Worker Settings

The value of Response Threshold θ_i is determined in a worker when the Worker process is activated. A uniform random number generator by C++11 standard libraries 32bit Mersenne Twister is reformed as a normal random number by a normal distribution generator. A normal random number takes a value in the range of 0 to 2 according to a normal distribution with $\mu = 1.0, \sigma^2 = 0.4$. The same uniform random number generator is also used to decide whether the task processing should be started or not.

5.2.4 Operation flow

When the Mediator receives a Job, it divides the Job into tasks immediately to transfer them to Stigmergy. After that, the Mediator requests a list of processed tasks from Stigmergy every 0.5 second, and checks the task-completion status.

Each Worker queries Stigmergy for the list of waiting tasks every second. The Worker decides whether or not

to process the task randomly. As soon as the task to be processed is decided, the Worker terminates checking the waiting list and gets the data of the task to be processed from Stigmergy. Finally, the Worker returns the results to Stigmergy when the processing is completed.

5.3 Experiment 1: Verify Auto Scaling

5.3.1 Experimental method

We prepared a bare system that does not apply Reaction Threshold model, namely **System-E1A**, and a system applied with the model as **System-E1B**. System-E1A includes 30 Workers while System-E1B includes 50 Workers. The parameters of System-E1B are set to be 10 Workers who do not operate on average when a single Job divided into 30 tasks is launched.

First, we observe the number of operating Workers and processing time when a single Job divided into 50 tasks is launched into both systems. Next, two Jobs divided into 50 were input to System-E1B, and we observed the number of active Workers.

5.3.2 Evaluation results for Ex.1

The processing was completed in about 20 seconds when a Job divided into 50 tasks was launched into System-E1A, and in about 26 seconds when it was launched in System-E1B.

The upper graph of Fig.6 shows the number of Workers processing a Task for every 0.1sec. Although few Workers start to process a Task soon after the Job is launched, 30 Workers are busy after about 10 seconds.

The lower graph of Fig.6 shows the transition in the number of active Workers when two Jobs are launched into System-E2B. After the processing is prolonged, the number of active Workers is still increasing with time: at about 20 seconds 35 Workers are active. Thus, the application of the Reaction Threshold model exhibits effects of Auto Scaling.

5.4 Experiment 2: Verify Setting Priority for each Task/Job

5.4.1 Experimental method

System-E2A and **System-E2B** consisting of the same 40 Workers are prepared for the next experiment. This experiment adopts the same function S as Equation (3), and we used the size of A_j as its priority.

Every second, we count the number of tasks completed for each Job, and verify feasibility of the processing speed according to the priority. Two Jobs divided into 40 Tasks are given to each system at the same time. In System-E2A, two Jobs are given with the same priority. In System-E2B, Job2 is given a priority A_2, which is 3 times larger than the priority A_1 of Job1.

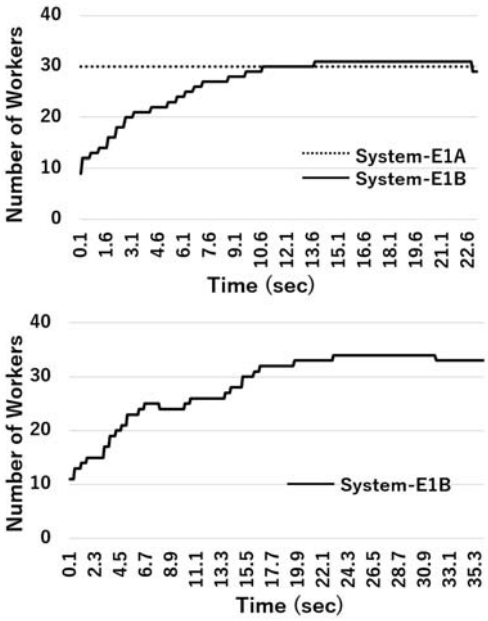

Fig. 6: Number of Workers which is processing task for each 0.1sec of experiment 1 (Upper: a graph when give a Job, Lower: a graph when give two Jobs)

5.4.2 Evaluation results for Ex.2

Fig.7 shows the results of Experiment 2. Two Jobs with the same priority are completed at almost the same time, whereas Jobs with a higher priority are completed earlier. It also shows that the number of Workers is allocated according to the ratio of A_j. This confirms that the Setting Priority can be realized without communicating between the server that controls the processing and the one that has the nodes.

5.5 Discussion

In addition to the features of Blackboard architecture and Data Flow architecture, we were able to acquire two functions of Auto Scaling and Setting Priority for each Task/Job. These features can be utilized as middleware for P2P distributed processing. By packaging data and programs and putting them on Stigmergy, a general-purpose distributed processing infrastructure can be realized without changing the workers. It is very easy to build and expand the system as there is no central computer that supervises the Workers, and each Worker can freely set its own parameters without negotiating with others. In addition, as the percentage of workers who are not active can be controlled by the parameter $\bar{T}$, it is also possible to adjust redundant systems and so forth in systems with large task-volume fluctuations. Finally, as it is also possible to assign priority to Jobs and tasks, the proposed architecture is highly useful as a base for general purpose distributed processing.

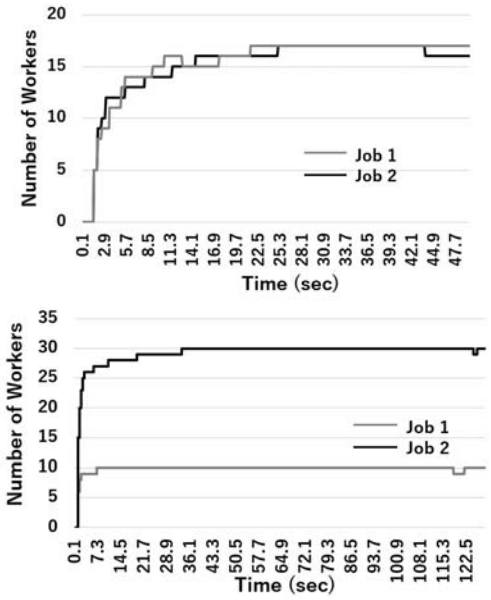

Fig. 7: Number of Workers which is processing task for each 0.1sec of experiment 2 (Upper: System-E2A, Lower: System-E2B)

6. Conclusions

This paper proposed a new model of autonomous decentralized system inspired by the ecology of social insects using the Response Threshold model based on the concept of Stigmergy. The model was implemented and experiments were conducted to verify its effectiveness.

From experiments, it is shown that our model realizes two new functions:

- Auto Scaling
- Setting Priority for each Task/Job

into the Stigmergy model, which is a well-known strategy for distributed systems design based on the Blackboard architecture.

References

[1] M. Dorigo, E. Bonabeau, and G. Theraulaz, "Ant algorithms and stigmergy," *Future Generation Computer Systems*, vol. 16, no. 8, pp. 851 – 871, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X0000042X

[2] E. Bonabeau, A. Sobkowski, G. Theraulaz, and J.-L. Deneubourg, "Adaptive Task Allocation Inspired by a Model of Division of Labor in Social Insects," Santa Fe Institute, Working Papers 98-01-004, Jan. 1998. [Online]. Available: https://ideas.repec.org/p/wop/safiwp/98-01-004.html

[3] Y. Ishii and E. Hasgeawa, "The mechanism underlying the regulation of work-related behaviors in the monomorphic ant, myrmica kotokui," *Journal of Ethology*, vol. 31, no. 1, pp. 61–69, Jan 2013. [Online]. Available: https://doi.org/10.1007/s10164-012-0349-6

[4] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, Apr 1997.

[5] T. Liao, K. Socha, M. A. M. de Oca, T. StÃijtzle, and M. Dorigo, "Ant colony optimization for mixed-variable optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 503–518, Aug 2014.

[6] G. Crina and A. Ajith, *Stigmergic Optimization: Inspiration, Technologies and Perspectives*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–24. [Online]. Available: https://doi.org/10.1007/978-3-540-34690-6_1

[7] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan 2003.

[8] K. Petersen, R. Nagpal, and J. Werfel, "Termes: An autonomous robotic system for three-dimensional collective construction," in *Robotics: Science and Systems*, 2011.

[9] H. F. Durrant-Whyte, N. Roy, J. Werfel, K. Petersen, and R. Nagpal, "Distributed multi-robot algorithms for the termes 3d collective construction system," 2011.

[10] A. Byrski, E. Å ‖ iderska, J. Å ‖ sisz, M. Kisiel-Dorohinicki, T. Lenaerts, D. Samson, B. Indurkhya, and A. NowÃl', "Socio-cognitively inspired ant colony optimization," *Journal of Computational Science*, vol. 21, pp. 397 – 406, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877750316302198

[11] T. Miyoshi, "Synthesijer," http://synthesijer.github.io/web/, access: Apr. 2016.

[12] E. Bonabeau, G. Theraulaz, and J.-L. Deneubourg, "Quantitative study of the fixed threshold model for the regulation of division of labor in insect societies," *Proc. Roy. Soc. London B 263*, pp. 1565–1569, 1996.

[13] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy, "The hearsay-ii speech-understanding system: Integrating knowledge to resolve uncertainty," *ACM Comput. Surv.*, vol. 12, no. 2, pp. 213–253, June 1980. [Online]. Available: http://doi.acm.org/10.1145/356810.356816

[14] K. Mori, H. Ihara, K. Kawano, M. Koizumi, M. Orimo, K. Nakai, H. Nakanishi, and Y. Suzuki, "Autonomous decentralized software structure and its application," in *Proceedings of 1986 ACM Fall Joint Computer Conference*, ser. ACM '86. Los Alamitos, CA, USA: IEEE Computer Society Press, 1986, pp. 1056–1063. [Online]. Available: http://dl.acm.org/citation.cfm?id=324493.325044

[15] K. Mori, "Autonomous decentralized systems: Concept, data field architecture and future trends," in *Proceedings ISAD 93: International Symposium on Autonomous Decentralized Systems*, 1993, pp. 28–34.

[16] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive load sharing in homogeneous distributed systems," *IEEE Trans. Softw. Eng.*, vol. 12, no. 5, pp. 662–675, May 1986. [Online]. Available: http://dl.acm.org/citation.cfm?id=5527.5535