

EVENT-DRIVEN LOAD BALANCING OF PARTIALLY REPLICATED OBJECTS THROUGH A SWARM OF MOBILE AGENTS

Sarah Abdul-Wahid, Răzvan Andonie*, Joseph Lemley, James Schwing, Jonathan Widger
Computer Science Department
Central Washington University
Ellensburg, USA

email: {abdulwahids, andonie, lemleyj, schwing, widgerj}@cwu.edu

ABSTRACT

Swarms of mobile software agents, imitating the behavior of insects, can solve complex tasks. Such agents individually have simple behavior. However, as a collective unit, constructive behavior emerges, as it does in insect colonies. The problem we address is optimizing the distribution of objects with partial replication over a computer network. We present a new approach, implemented as a prototype, based on swarm intelligence. The system performs dynamic, event-driven, distributed load balancing. Every node of the network is capable of producing new events and introducing them into the network for computation. The architecture is general and can be used as a support for clustering and task allocation applications.

KEY WORDS

Swarm intelligence, mobile intelligent agents, distributed dynamic load balancing.

1 Introduction

There is no universally accepted definition to describe agents. For instance, according to Elamy [1], an agent is a software entity which possesses special intelligent properties, such as autonomy, sociability, reactivity, proactivity, temporal continuity, learning ability, reasoning, veracity, bio-induction, mobility, cooperation, and negotiation. Such intelligent properties allow agents to initiate their actions without the need of direct intervention by humans or other entities. In this way, they are able to interactively cooperate and communicate with each other and with their environment, to accomplish special tasks that cannot be performed by conventional software. The environment has a certain level of autonomy with respect to what agents do, but it can also be influenced by the agent's behavior.

A *mobile* agent can migrate from machine to machine in a heterogeneous network. On each machine, the agent interacts with the environment and other resources to accomplish its task. The machine specific environment can be considered as a stationary agent. A mobile agent can choose different migration strategies depending on its task and current network conditions, and then change its strate-

gies as network conditions change [2]. Such agents are particularly attractive in distributed information-retrieval applications on the Internet [3]. By moving to the location of an information resource, the agent can search the resource locally, eliminating the transfer of intermediate results across the network and reducing end-to-end latency. This may be also considered in a mobile-computing scenario, where users have portable computing devices with only intermittent, low-bandwidth connections to the main network [4].

An agent's behavior is the result of simple rules based on local interactions. A fundamental inspiration of the agent computing paradigm is a swarm of insects. Swarm intelligence [5, 6] is intrinsically a bottom-up approach. Through the cumulative action of agents, constructive behavior emerges. This is similar to what happens in insect colonies that create complex social behavior and structures from the combined efforts of individuals with extremely limited intelligence. There is much recent interest in the synthesis of bio-inspired swarming behavior for engineered systems.

A swarm-type system consists of a finite collection of agents, each of which has fairly limited capabilities on its own. Interactions between agents can arise through direct or indirect communication; two agents interact indirectly when one of them modifies the environment and the other responds to the new environment at a later time. By exploiting such local communication forms, agents can detect changes in the environment dynamically [6]. Thus, agents can *autonomously adapt* their own behavior to the new changes. Swarm systems emphasize *self-organization* capabilities. The intelligence exhibited is not present in the individuals, but rather emerges out of the entire swarm. Swarm-type systems are *robust*, *redundant*, and *flexible*. No central coordination takes place, which means that there is no single point of failure. No single agent is essential, that is, it can be dynamically added or removed.

One group of swarm-type system applications is data clustering, most inspired by the work of Deneubourg *et al.* [7]: Mobile agents, such as ants, carry objects with the goal of grouping similar objects [8, 9, 10]. The ants can be endowed with a short-term memory. An overview of proposed ant clustering algorithms can be found in [10]. Recent applications include Web mining [21, 11]. These clus-

*Corresponding author. Authors are listed in alphabetical order.

tering procedures use batch learning and the initial clustering conditions are stable. A different swarm-type model, described by Bonabeau *et al.* [12], has inspired distributed task allocation applications [19, 20].

A few attempts have been made to provide general purpose dynamic distributed swarm-type systems. In [23], agents have more autonomy and can define their own migration policy as well as the migration of other agents. The BISON¹ project explores the possibility of constructing distributed, self-organizing information systems as ensembles of agents that mimic the behavior of biological processes. Anthill is a toolbox implemented during this project [13]. An Anthill system is composed of a collection of interconnected nests. Each nest is a peer entity that makes its storage and computational resources available to swarms of ants that travel across the network trying to satisfy user requests. During their life, ants interact with services provided by visited nests, such as storage management and ant scheduling.

Like Anthill, the model we introduce in this paper is a self-organizing swarm-type completely distributed system with dynamic load balancing capability. However, our problem statement and design principles differ from Anthill. In our problem statement, we distribute objects on a computer network. Objects may be partially replicated on different nodes, each node containing unique objects. This not only makes the system robust to individual node failures but also increases performance in the distributed environment. Applications of data replication are distributed databases [14], dynamic data replication [15], and data migration in dynamic content web servers [16]. Our contribution is a general tool which performs a dynamic, event-driven, distributed load balancing. From each node, a swarm of agents move objects to the other nodes, performing the load balancing. There is no central node and no single agent is essential, since data is replicated. The partial replication factor (the minimum number of copies kept by the system and distributed to different nodes) is a parameter of the system and gives the minimum degree of redundancy. Superfluous copies not accessed over time are deleted. The system is event-driven: Every node of the network is capable of producing new events. Nodes can be added and eliminated dynamically. Our platform prototype can embed distributed dynamic clustering/task allocation applications such as dynamic distributed databases. Our mobile agents are bio-inspired, very similar to the red harvester ants (*Pogonomyrmex barbatus*) described in [17, 18]. Thus we have named them *Pogo ants*.

2 An Overview of Our Algorithm

We summarize the behavior of social insects, such as Pogo ants that Gordon describes in [18], as follows. Pogo ant colonies operate with no central control and no ant directs the behavior of another. The numbers of ants engaged in

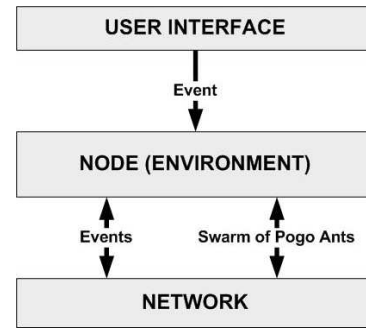


Figure 1. Data Flow.

any task is determined by current environmental conditions and colony needs, similar to task allocation in a distributed computer system.

Our algorithm is designed to capture the essence of this description. The components of the algorithm are the environment (tied to a node in the system), the Pogo ants (intelligent agents), and the cells (containers for objects like data, tasks, or events). The environments are stationary, whereas the ants are mobile. The ants, serving as containers for cells, move from environment to environment, following simple rules of behavior. External events drive this behavior in various directions. The ants possess short-term memory of their connection to nodes requiring the cells they carry. The collective result of these behaviors is event-driven. Each event triggers dynamically the redistribution of partially replicated objects; this dynamic load-balancing scheme runs with lower priority in the background. The remainder of this section further details these components and their interactions.

Adaptive Environment. A single environment object exists on each node of the network. The state of the environment changes, but its position is fixed. Pogo ants exist within the environment. The environment interacts with the ants within its borders and with ants requesting migration to it. The environment serves as the intermediary for events, as shown in Fig. 1.

Connecting Cells to Nodes on the Network. Each cell has a short term memory of nodes which have interacted with it. The memory decays slowly over time, analogous to the concept of pheromone dissipation. Eventually, a cell can “forget” that it has ever interacted with a particular node.

The Event. An event can be initiated from any node on the network. Each event is assigned a unique identifier which includes the identifier of the node initiating it. The event is broadcast to all environments. Each environment processes the event in parallel and returns all results to the node that initiated it.

Defining “Similar Cells”. Two cells are *similar* if they have been accessed by the same environment at the same rate, plus or minus a given threshold value.

Mobile, Adaptive, Autonomous Agents (Pogo Ants) as

¹<http://www.cs.unibo.it/bison>

Containers for Cells. Cells do not exist “unattached” in the environment; they exist only within the framework of Pogo ants. Each ant may contain one or more cells. A colony of ants exists in each environment (Fig. 1). Ants can migrate to other environments, taking with them the cells they contain. Ant operations include join and split. An ant can interact with another ant carrying similar cells by joining. Following the join operation, one of the ants now contains all the cells and the other ceases to exist. Thus, similar cells are grouped together in a single ant and the number of ants is reduced. If an ant contains cells that are not similar, the ant can split into as many as three ants. In this process, the cells are assigned to new ants according to their measure of similarity. The split operation increases the number of ants. Thus, as event patterns change, cells can be regrouped according to their measure of similarity and in the process the state of the colony changes.

Determining an Ant’s “Fitness” for an Environment. Each cell a Pogo ant carries may have been requested in some event by one or more environments. The average access counter is the average of the number of times all cells within an ant have been accessed by the ant’s current environment. A *hunger* value is computed as the inverse of the average access counter. A low hunger value indicates that the ant *fits* well in its current environment. If none of the cells an ant carries have been accessed by the current environment, the hunger value is set at the maximum integer value. If the ant migrates to a new environment, the hunger value is automatically recalculated from the perspective of the new environment.

Generating and Distributing “Food” for the Pogo Ants. Food is an attribute of the environment. A Pogo ant requires food in order to be ready to join with another ant. Food within an environment is generated when the environment initiates an event. The amount of food generated with each event is determined by the current state of the environment. Before distributing the food, the ants are sorted, in nondecreasing order, according to their hunger value. The available food is distributed according to this ordering scheme. This is analogous to “survival of the fittest.” It is possible that there is not enough food for all ants within the environment. The ants that have been fed are marked as such. The ants that have not received food are marked as *unfed*. The fed ants now seek to join with another ant within the environment. They search for the most compatible union.

The Signal to Swarm – Dynamic Load Balancing. Following distribution of food and execution of the join operation, the environment signals all Pogo ants within its colony to swarm. The ants cannot ignore this signal. Each ant broadcasts a request to migrate to all other environments on the network. The environment receiving the request examines its current state and determines this new ant’s relative fitness. The request is granted only if the new ant is a good fit. It is possible that more than one environment grants a particular ant’s request to migrate. This results in replication of the cells carried by the ant. If an ant

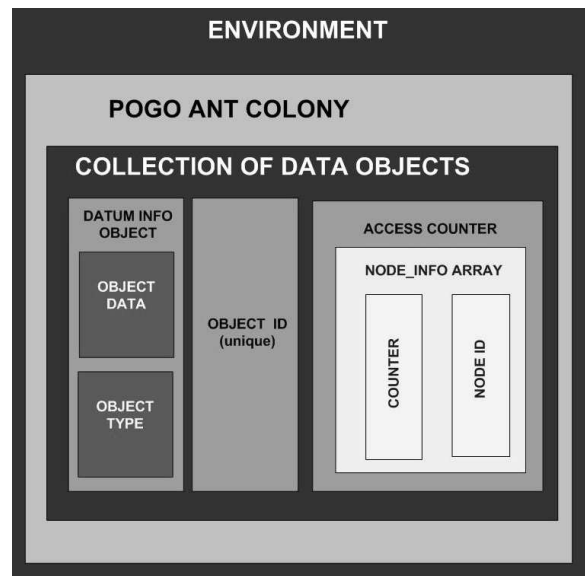


Figure 2. Environment containing the Pogo ants colony. Each ant contains a collection of data objects.

does not fit well into any environment, it does not migrate.

Separating Dissimilar Cells. Following migration, there might be unfed Pogo ants that were forced to remain in their current environment. If an unfed ant who remained contains more than one cell, it can split into as many as three ants. Its cells are assigned to the new ant according to its measure of fitness for the current environment. In this manner, dissimilar cells can be separated.

Partial Replication. The primary advantage of distributing objects on a P2P network is the the partial replication of data. This provides data security in a flexible manner. Even if a cell is never required by an event, the number of copies defined by the partial replication factor is guaranteed across the network. Only one copy exists on any given environment. If a cell is required often by numerous environments, more than the minimum number of copies exists across the network. If event requirements change over time, and the cell is no longer frequently required, superfluous copies are deleted until the minimum number is restored.

3 Implementation Details

The algorithm was implemented in C# using the TCP/IP communication protocol. Details of the major components follow.

Environment Class. Environment attributes include the following: Pogo ants, events, results of the current event, the amount of food currently in the environment, the rate of dissipation of a cell’s access counter, the join threshold, the split threshold, the various parameters of the food generation function, and the partial replication factor.

Pogo Ant Class. Attributes of the Pogo ant class in-

clude the cells it carries, the hunger value, the identifier of the ant's current environment, the average of the cells' access counters with respect to the current environment, and a Boolean value indicating whether it is fed during the learning process. The hunger value is the inverse of the average access counter value; if the average access counter is zero, hunger is set at the maximum integer value.

Cell Class. The cell class contains a Datum object, an AccessCounter object, and its unique identifier.

Connected Nodes Tracking Method. A standalone tracker program determines which nodes are running the program, and assigns unique identifiers to each available node. If a node disconnects, it is removed from the list and its cells are "lost." This fact emphasizes the importance of partial data replication. If a new node connects to the system, it will eventually be populated with cells as the algorithm runs and adjusts to the changed state.

Distribution. When the program is initiated, the tracker determines which nodes are available. Each cell is assigned to a single Pogo ant. The ants are assigned to nodes based on the cyclic (interleaved) allocation scheme. If there are p available nodes, the ant is assigned to the node with identifier equaling the ant's index $\mathbf{mod} p$.

The Link Connecting Cells to Nodes. In addition to the datum and the unique identifier, each cell contains an AccessCounter. This stores the number of times each node on the network has required this cell during an event, thus implementing a bond between each cell and one or more nodes on the network. These connections are volatile. When a cell is accessed during an event, its node specific access counter is incremented, strengthening the bond. After every event, the access counters of all cells are slightly decremented, thus weakening all connections. After sufficient time, if a cell is not required by a particular node, connection to that node disappears.

The Learning Process. Environments and Pogo ants are adaptive. This learning process involves the following: the generation and distribution of food, the join operation, the migrate operation, the split operation, and the deletion of superfluous copies of cells. The learning sequence is initiated at regular intervals, for example after every q events. Internal events, like maintenance or operating system tasks, do not trigger learning. Only events explicitly addressed by the user, like tasks or queries, are effective. For example, in order to delete superfluous copies of a cell, queries must be done to determine how many copies of the cell exist on the network. These queries do not affect food generation. They are not counted when deciding when to perform the learning process, and the access counters of cells accessed in this process are not incremented.

Food Generation and Distribution. While in learning mode, an event generates food within an environment. The quantity produced is determined by the number of cells within the environment's colony, the number of results obtained by the event, and several parameters according to formula:

$$fg = A^{(optn - \#cells)/B} - C / (maxn - \#cells)$$

The value fg is the food generated by a specific event which is added to the environment's current food value. The $optn$ is the environments' optimum number of cells, $maxn$ is the maximum number of cells for the environment's colony (the node's storage limit), and $\#cells$ is the current number of cells in the colony. The values A , B , and C are parameters of the problem. The value of $maxn$ is enforced by the system and $\#cells$ is always less than $maxn$.

The environment distributes the food to the Pogo ants in its colony. The ants are first sorted, in non-decreasing order, according to their measure of fitness (the hunger value). The most fit ants receive food first. The amount of food required by an ant is determined by the following formula: $requiredFood = hunger * \#cells$.

The term *hunger* is the measure of fitness (smaller is better) of the ant for the environment, and $\#cells$ is the number of cells carried by the ant. Thus, the fittest ants require less food per cell and are more likely to be fed. The ants are fed until the food is consumed, or until each ant has received its required amount of food. If there is not enough food to feed an ant, it is marked as being unfed.

Regrouping Similar Cells. Following the feeding process, each fed Pogo ant searches all ants in its colony in order to join with a compatible ant. For a match to be compatible, the two hunger values must be within a certain threshold value. The most compatible match is chosen. Following the join operation, all cells of both ants exist in one of the ants while the other ant ceases to exist. If no compatible match is found, the ant does not join with another.

Redistributing Cells Across the Network. Following the join process, the environment signals its colony to swarm. Each Pogo ant broadcasts a request to migrate to all other environments. The hunger value of the migrating ant is calculated with respect to its prospective new environment. If this is less than the hunger value of the environment's "least well fed ant," the incoming ant is accepted. If no environment accepts the ant's request to migrate, the ant does not migrate. If multiple environments accept the request to migrate, the result is that several copies of the ant are made, and there is replication of frequently required information.

Separating Dissimilar Cells. Following the migration operation, there might be unfed Pogo ants that had to remain in their current environment. These ants then call the split method if they contain more than one cell. If so, the ant will classify cells as follows: *i*) full cells, *ii*) hungry cells, and *iii*) "just right" cells, by comparing the access counter of each cell to the average access counter of the ant, plus or minus the split threshold value. Each of the resulting groups defines the cells for a new ant, and the original ant ceases to exist.

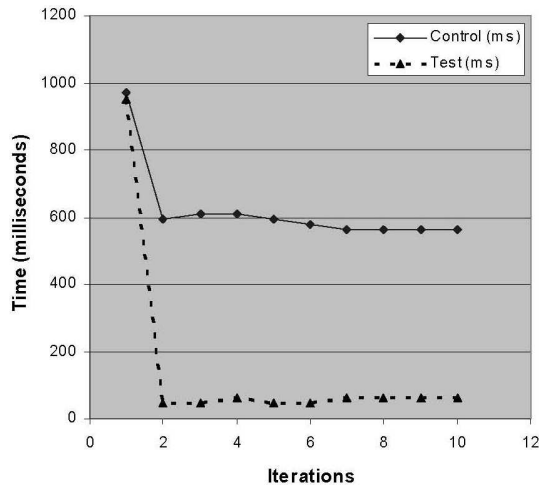


Figure 3. Time Benchmark.

Deletion of Superfluous Cells. Following the split method, if an unfed ant contains only one cell whose replication factor is met by other environments, then the ant and its cell are deleted.

4 Preliminary Results

Our network for experiments consists of 10 workstations running Microsoft Windows XP SP2 interconnected via 100 MBPS Ethernet running the Pogo ant program. In these experiments, the distributed objects are units of data in a database; events are queries for this data. In order to simulate distribution of Pogo ants across a very large network, such as the Internet, a lag of 100 ms is added by the program for all communication between nodes. A preliminary experiment was designed to measure the improvement in response time provided by the learning methods. We began with 58 Pogo ants and 58 cells on the network, each ant containing a single cell. The ants were distributed according to the interleaved allocation method. To test grouping behavior, we repeatedly submitted a single query from only one node, say i . This node initially contained five ants, each containing one cell. The query was repeated ten times.

We did two experiments: one with learning disabled (control), and one with learning enabled (test). The response times were measured and grouping behavior was observed. In the control experiment, node i maintained its initial population of cells. In the test experiment, however, the node's state changed.

The four cells required to complete the query were not initially on node i . This explains the long response time for the initial query in both experiments. In the test experiment, learning occurred after the initial iteration and the ants containing the cells required for completion of the query migrated to node i ; the response time decreased by a factor of ten. After completion of iteration number 8, the four cells which were carried to node i by four ants were

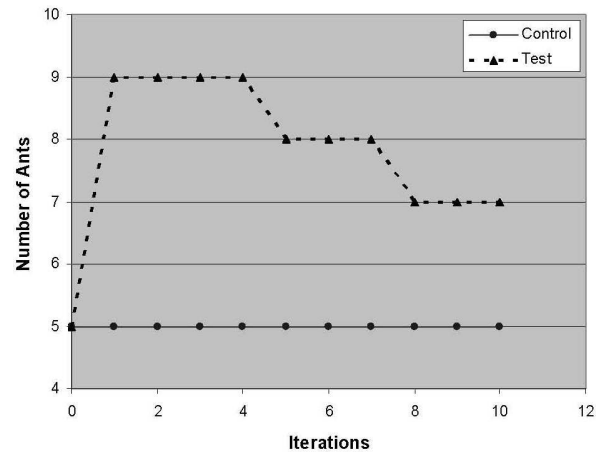


Figure 4. Grouping of Pogo Ants on Node i .

clustered into 2 ants: the three most similar cells were in one ant while the fourth cell was held by the second ant. In the control experiment, no migration occurred and response time remained nearly constant. The benchmark results are displayed in Fig. 3 and Fig. 4.

5 Conclusion

We have designed a self-organizing swarm-type system of mobile agents, called Pogo ants, as a support for completely distributed applications. The system performs a dynamic, event-driven, distributed load balancing. Every node of the network is capable of producing new events and introducing them into the network for computation. The architecture is general and can be used as a support for clustering and task allocation applications. Our preliminary tests confirmed the predictions. More complex tests involving simultaneous events initiated from multiple nodes are in process.

For future work, we plan to investigate defining bio-inspired, distributed-computing design patterns [22] by generalizing the Pogo ant operations together with search and load-balancing procedures.

An interesting result reported in [8] is that a swarm of heterogeneous ants (ants with diverse parameters) can perform better than a homogeneous swarm. Meanwhile, dynamically switching the behavioral pattern of the agents may also improve the performance of the clustering algorithm. We intend to investigate both these aspects in our future experiments.

References

- [1] A. H. Elamy, Perspectives in agent-based technology, *AgentLink News Newsletter*, European Co-ordination Action for Agent Based Computing, Issue 18, August 2005, 19–22. Available at <http://www.agentlink.org/>.

- [2] B. B. Brewington, R. Gray, K. Moizumi, Mobile Agents for distributed information retrieval, in M. Klusch (Ed.) *Intelligent Information Agents* (Springer-Verlag, 1999) 355–395.
- [3] D. Kotz, R. S. Gray, Mobile agents and the future of the Internet, *ACM Operating Systems Review*, 33(3), ACM Press, 1999, 7–13.
- [4] R. S. Gray, G. Cybenko, D. Kotz, R. A. Peterson, D. Rus, D'Agents: application and performance of a mobile-agent system, *Software — Practice and Experience*, 32(6), 2002, 543–573.
- [5] E. Bonabeau, M. Dorigo, G. Theraulaz, *Swarm intelligence: from natural to artificial systems* (Santa Fe Institute in the Sciences of the Complexity, Oxford University Press, New York, Oxford, 1999).
- [6] M. Bedal, J. Gaber, H. El-Sayed, A. Almojel, Swarm intelligence, in S. Olariu, A. Y. Zomaya (Eds.) *Handbook of Bioinspired Algorithms* (Chapman & Hall/CRC, 2006) 55–84.
- [7] J. L. Deneubourg, S. Gross, N. R. Franks, A. Sendova-Franks, C. Detrain, L. Chrétien, The dynamics of collective sorting: robot-like ants and ant-like robots, *Proceedings of The First International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, 1991, Paris, France, MIT Press, Cambridge, MA, USA, 356–363.
- [8] E. D. Lumer, B. Faieta, Diversity and adaptation in populations of clustering ants, *Proceedings of The Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, 1994, Brighton, U. K., MIT Press, Cambridge, MA, USA, 501–508.
- [9] V. Hartmann, Evolving agent swarms for clustering and sorting, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005)*, Washington DC, USA, ACM, 2005, 217–224.
- [10] A. L. Vazine, L. N. de Castro, E. R. Hruschka, R. R. Gudwin, Towards improving clustering ants: an adaptive ant clustering algorithm, *Informatica*, 29(2), 2005, 143–154.
- [11] A. Abraham, V. Ramos, Web usage mining using artificial ant colony clustering and genetic programming, in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC2003)*, Australia, IEEE Press, 2003, 1384–1391.
- [12] E. Bonabeau, A. Sobrowski, G. Theraulaz, J. L. Deneubourg, Adaptive task allocation inspired by a model of division of labour in social insects, in *Santa Fe Institute Working Paper 98-01-004* (Santa Fe Institute, 1998).
- [13] A. Montresor, H. Meling, Ö. Babaoğlu, Messor: load-balancing through a swarm of autonomous agents, in G. Moro, M. Koubarakis (Eds.) *Agents and Peer-to-Peer Computing*, (Lecture Notes in Artificial Intelligence, volume 2530, Springer-Verlag, Berlin, 2003) 125–137.
- [14] M. Nicola, M. Jarke, Performance modeling of distributed and replicated Databases, *IEEE Transactions on Knowledge and Data Engineering*, 12(4), 2000, 645–672.
- [15] R. Christodoulopoulou, R. Azimi, A. Bilas, Dynamic data replication: an approach to providing fault-tolerant shared memory clusters, in *Proceedings of the Ninth Annual Symposium on High Performance Computer Architecture (HPCA-9'03)*, 2003, 203–214.
- [16] G. Soundararajan, C. Amza, On-line data migration for autonomic provisioning of databases in dynamic content web servers, in *Proceedings of the 2005 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON '05)*, Toronto, Ontario, Canada, IBM Press, 2005, 268–282.
- [17] D. M. Gordon, *Ants at work: how an insect society is organized* (Free Press, New York, 1999).
- [18] D. M. Gordon, The organization of work in social insects colonies, *Complexity*, 8(1), 2003, 43–46.
- [19] K. H. Low, W. K. Leow, H. Marcelo, Task allocation via self-organizing swarm coalitions in distributed mobile sensor network, in *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04)*, 2004, 28–33.
- [20] M. D. Peysakhov, W. C. Regli, Ant inspired server population management in a service based computing environment, in *Proceedings of the IEEE Swarm Intelligence Symposium (SIS 2005)*, Pasadena, California, 2005, 357–364.
- [21] J. Handl, B. Meyer, Improved ant-based clustering and sorting in a document retrieval interface, in J. J. Merelo, J. L. F. Villacañas, H. G. Beyer, P. Adams (Eds.) *Proceedings of 7th International Conference on Parallel Problem Solving from Nature*, Granada, Spain, 2002.
- [22] Ö. Babaoğlu, G. Canright, A. Deutsch, G. A. Di Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, Design patterns from biology for distributed computing, in *ACM Transactions on Autonomous Adaptive Systems*, to appear, 2006.
- [23] I. Satoh, Bio-inspired deployment of distributed applications, in *Proceedings of the 7th Pacific Rim International Workshop on Multi-Agents (PRIMA 2004)*, Auckland, New Zealand, Springer-Verlag, Lecture Notes in Computer Science, vol. 3371, 2004, 243–258.