Adaptive Distributed Database Replication Through Colonies of Pogo Ants

Sarah Abdul-Wahid, Răzvan Andonie, Joseph Lemley, James Schwing, and Jonathan Widger

Central Washington University
Computer Science Department
Ellensburg, WA 98926, USA
abdulwahids@gmail.com, andonie@cwu.edu,
joelemley@gmail.com, schwing@cwu.edu, jon.widger@gmail.com

Abstract

We address the problem of optimizing the distribution of partially replicated databases over a computer network. Replication is used to increase data availability in the presence of site or communication failures and to decrease retrieval costs by local access if possible. We present a new bio-inspired replication management approach which is adaptive, completely decentralized, and based on swarm intelligence. Each node has the autonomy to start at any time, depending on the internal state of its stored data objects, a redistribution process. "Redistribution" means replicate, create, delete, update, or move data objects to other nodes of the network. The redistribution process is a dynamic load-balancing scheme which runs with lower priority in the background. The system is event-driven, but the learning process is not synchronized with the events.

1 Introduction

Data replication consists of maintaining multiple copies of data, called replicas, on separate computers. Replication improves availability by allowing access to the data even when some of the replicas are unavailable. It also improves performance by the following: *i)* reducing latency, since users can access nearby replicas, thus avoiding remote network access; and *ii)* increasing throughput, since multiple computers can serve the data simultaneously. Replication management in distributed database systems concerns the decision of when and where to allocate physical copies of logical data fragments and when and how to update them to maintain an acceptable degree of mutual consistency. In this paper, we focus on adaptive data placement, in which each node maintains statistics about the query workload and automatically moves data and establishes copies of data at different nodes in order to adjust the data placement to the current demand. We approach a solution to this problem using swarm intelligence.

Swarm intelligence [6, 8], 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 having extremely limited intelligence. There is much recent interest in the synthesis of bio-inspired swarming behavior for engineered systems. Colonies of mobile software agents can solve complex tasks, including distributed dynamic load balancing. Such a 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 mechanisms, agents can detect changes in the environment dynamically [6]. Thus, agents can autonomously adapt their own behavior to the new changes. The intelligence exhibited is not present in the individuals, but rather emerges out of the entire colony. The colony is self-organizing, robust, redundant, and flexible. No central coordination takes place, which means that there is no single point of failure. No single agent is essential and therefore can be dynamically added or removed.

Our paper describes a novel distributed data replication management system. Replicas of data objects are created dynamically on nodes which frequently access these objects, in order to minimize inter-node communications and response time. Conversely, these replicas are deleted after being idle for a period of time. The system ensures a minimum number of replicas, thus making the database resistent to node failures. In addition, nodes can be dynami-

Authors are listed in alphabetical order. Răzvan Andonie is the corresponding author.

1-4244-0910-1/07/$20.00 ©2007 IEEE

cally added. They are then populated with data as the need arises. The system is based on swarm intelligence and completely decentralized. It can improve its performance by dynamic load balancing.

We use a partially replicated distributed database to evaluate our system. The database is distributed over a P2P network. Transactions (which can be queries or updates) from any node trigger events. Events change the internal states of the accessed data objects. In our experiments, the transactions are queries.

After describing in Section 2 how our contribution relates to previous work, in Section 3 we introduce our partial replication scheme. Section 4 describes our load balancing scheme. In Section 5 we explain how the database is generated and queried. Sections 6 and 7 contain the preliminary results and the conclusions.

2 Related Work

2.1 Work related to data replication

There are several possible replication models considered in the literature [10, 16–19, 24, 26, 37]; some of these are dynamic. The problem of finding an optimal replication scheme in a general network (i.e., a replication scheme that has a minimum cost for a given read-write pattern), has been shown to be NP-complete for the static case [5, 18, 37, 38]. For this reason it is unlikely to find an efficient and convergent-optimal dynamic allocation algorithm.

The synchronization and communication required to create and update the replicas introduces a delay when operations are performed. In time-constrained systems, or systems distributed over a bandwidth-constrained area, such operational delays generally prove unacceptable. Asynchronous data replication is commonly used to mitigate these delays [4].

2.2 Work related to swarm-type distributed systems

One group of swarm-type system applications is data clustering, most inspired by the work of Deneubourg *et al.* [11]: Mobile agents, such as ants, carry objects with the goal of grouping similar objects [15, 22, 35]. The ants can be endowed with a short-term memory. An overview of proposed ant clustering algorithms can be found in [35]. Recent applications include Web mining [3,14]. These clustering procedures use batch learning and the initial clustering conditions are stable. A different swarm-type model, described by Bonabeau *et al.* [9], has inspired distributed task allocation applications [20, 29].

Some progress has been made to provide general purpose dynamic distributed swarm-type systems. Among these are BISON and the Bio-Networking Architecture (BNA). The BISON project (http://www.cs.unibo.it/bison), explores the possibility of constructing distributed, self-organizing information systems as ensembles of agents that mimic the behavior of biological processes. Anthill and Messor are toolboxes implemented during this project [23]. 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 traveling across the network to satisfy user requests. During their life, ants interact with services provided by visited nests, such as storage management and ant scheduling. The computational power offered by a network of Anthill nests is exploited by Messor, a grid computing system, by assigning a set of jobs comprising a computation to a dispersed set of nests. To determine how to balance the load among the computing nodes, Messor uses a bio-inspired algorithm. Messor ants drop objects they are carrying only after having wandered about randomly "for a while" without encountering object concentrations. Colonies of such Messor ants try to disperse objects (more specifically, jobs) uniformly over their environment, rather than clustering them into piles. Messor is completely decentralized, allowing every node in the system to generate new jobs and submit them to the network.

The BNA system [34, 36] applies key concepts and mechanisms in biological systems to design network applications. A BNA application is a group of distributed, autonomous, and diverse objects called cyber-entities (CEs). Each CE implements a functional service related to the application and follows simple behaviors similar to biological entities, such as reproduction, death, migration, and environment sensing. Different CEs may implement different behavior policies. Each CE may store and expend energy for living. CEs may gain energy in exchange for performing a service, and they may pay energy to receive a service from other CEs and to use network and computing resources. The abundance or scarcity of stored energy may affect various behaviors of a CE. For example, an abundance of stored energy is an indication of higher demand for the CE; thus, the CE may be designed to favor reproduction in response to higher levels of stored energy. A scarcity of stored energy (an indication of lack of demand or ineffective behaviors) may eventually cause the CEs death. In addition, CEs can implement evolutionary adaptation mechanisms, such as selection, crossover, and mutation [25].

In a previous paper [1], we have introduced a middleware platform 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 system is event-driven: Every node of the network is capable of producing

new events. Nodes can be added and eliminated dynamically. Our mobile agents are similar to the red harvester ants, *Pogonomyrmex barbatus*, described in [12, 13]. For this reason we have named them "Pogo" ants.

2.3 Relation to our present work

A good survey of replicated database systems can be found in [26]. The partially replicated database scheme we introduce here differs from these and from those described in [10, 16–19, 24, 37]. In our model, the number of replicas varies with conditions, but never goes below a certain level. To our knowledge, a swarm intelligence approach has not previously been applied to database replication.

Our present work is an enhancement of the Pogo ants software platform [1], which we have now customized for distributed database systems. We introduce here a fully functional partially replicated database and explore grouping behaviour of the replication management tool as complex event interactions occur.

There are obvious similarities between the life cycle of Pogo ants and CEs. As in all biological entities, they initialize, reproduce, migrate and die. In addition, the energy in a CE is similar to the hunger in a Pogo ant. However, our platform and BISON/BNA are different. In our approach, 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. In contrast, the BISON and BNA systems do not deal with object replication.

3 An adaptive partial replication scheme

We introduce a new replication model. Each data object is physically replicated *at least* $r \in \{1, 2, ..., n\}$ times on different nodes, where n is the number of nodes in the network. A value of $r = 1$ does not necessarily mean "no replication," whereas $r = n$ means "full replication." The number of nodes that can be safely eliminated from the system without information loss is $r - 1$. Replicas of data objects are created dynamically on nodes which frequently access these objects, in order to minimize inter-node communications and response time. Conversely, these replicas are deleted after not being accessed for a period of time. It is not assumed that the replicas are distributed evenly across the sites and it is undefined which copies are placed on which site, so that different degrees of quality of a replication schema can be modeled. Each node has knowledge only of its local state; it requires no knowledge of where data replicas exist.

The system is completely decentralized. Transactions (which can be queries or updates) from any node trigger events. Events change the internal states of the accessed data objects. Each node has the autonomy to start the redistribution process at any time, depending on the internal state of its stored data objects. "Redistribution" means create, delete, update, or move data objects to other nodes of the network. The redistribution process is a dynamic load-balancing scheme which runs with lower priority in the background. The system is event-driven, but the learning process is not synchronized with the events (i.e., we use asynchronous replication.) It has the ability to dynamically adjust to changing requirements based on a short-memory mechanism.

4 The swarm intelligence approach for load balancing

We summarize the behavior of social insects such as Pogo ants as follows. Pogo ant colonies operate without central control; no ant directs the behavior of another. The number of ants engaged in any task is determined by current environmental conditions and colony needs. Task allocation is the process of adjusting the number of ants engaged in each task according to varying conditions.

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), Pogo ants (intelligent agents), and cells (containers for objects such as 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, with dynamic load balancing of the replicated objects. 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, triggered by a transaction. Each event is assigned a unique identifier which includes the identifier of the node initiating it. The event is broadcast to all envi-

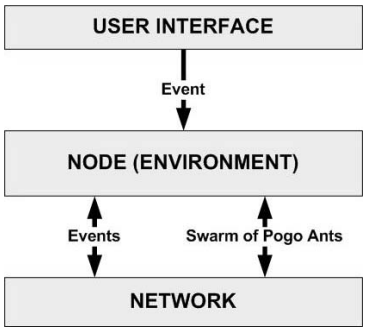

Figure 1. Data Flow.

ronments, which process the event in parallel and return all results to the node that initiated it.

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 Containers for Cells. Cells do not exist "unattached" in the environment; they exist only within the framework of Pogo ants. Each ant may hold one or more cells. A colony of ants exists in each environment. Ants can send replicas 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. In the join process the two ants merge into one ant. Thus, similar cells are grouped together in a single ant and the number of ants is reduced. If an ant holds dissimilar cells, 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 similarity. Thus, the state of the colony changes.

Determining an Ant's Fitness for an Environment. Each cell a Pogo ant carries may have been requested 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 an ant replica 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 before it can search for another ant to join with. Food within an environment is generated when the environment initiates an event. The amount of food generated with each event 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:

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

The value $foodGen$ is the food generated by a specific event, where:

- $optn$ is the environment's optimum number of cells.
- $maxn$ is the maximum number of cells for the environment's colony (the node's storage limit); this value is enforced by the system.
- $\#cells$ is the current number of cells in the colony and is always less than $maxn$.
- A, B, and C are parameters of the problem.

The food generated from each event is added to the environment's current food value. 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, analogous to "survival of the fittest." 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. 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 send a clone (replica) of itself to all other environments on the network. The environment receiving the request examines its current state and determines the relative fitness of the incoming ant. The request is granted only if this ant is a good fit. When an ant replica migrates, it does so as a complete entity, carrying all its cells. It is possible that more than one environment grants a particular ant's request. This results in multiple replication of the ant, and thus multiple replication of all cells carried by the ant. If an ant does not fit well into any environment,

STUDENTS				
Row	StudentId	Name	Year	Major
1	19	Eric	4	CS
2	89	Mariam	1	MATH
3	24	David	2	CHEM
4	45	Barbara	4	CS

CLASS		
Row	ClassName	StudentId
1	CS481	19
2	CS312	45
3	MATH273	89
4	CS481	45

Figure 2. Database of Two Tables.

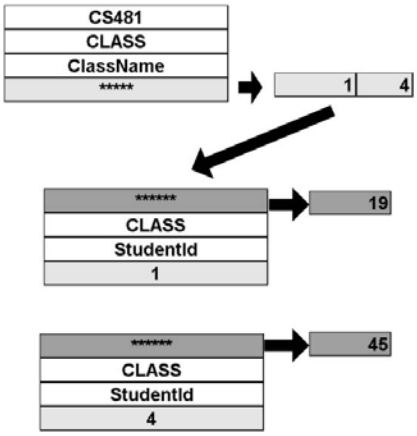

Figure 3. Two-step Query Process.

it is not replicated. Cells in excess of the minimum replication factor may be deleted, depending on their state. Ants without cells are deleted. This event driven swarming accomplishes dynamic load balancing. The current state of the environment (including its current load) determines the course of action of each ant in the environment: whether it is replicated or deleted.

Separating Dissimilar Cells. There may be unfed Pogo ants that were not able to send replicas. If any of these holds more than one cell, it can divide into up to three ants. In this manner, dissimilar cells can be separated.

5 The Replicated Database System

5.1 Database Decomposition

To maintain flexibility, a database, such as the one illustrated in Fig. 2, is decomposed to the cellular level. Each cell is an object which contains the piece of datum along with its relational database information. This relational information includes table name, column heading, and row number. Thus, the two tables from Fig. 2 decompose to a total of 16+8 cells. The cells of the database are distributed over all nodes currently on the network.

With this decomposition scheme, a query is accomplished by a two-step process. In the first step, the row number associated with a particular table's column heading is identified; in the second step, the datum value in this row with the appropriate second column heading is located. These steps are accomplished using query cells, which are a type of cell with one of the fields serving as a wildcard. A query which returns the id number of students who are CS majors could be written as follows:

> SELECT StudentId FROM CLASS WHERE ClassName EQUAL CS481.

The first step of the query is to identify the row numbers where the ClassName column datum value is CS481. After this is known, the datum value of the StudentId column in these rows can be ascertained. This process is depicted in Fig. 3. SQL style formatting of the query is used. Joins of tables can be accomplished with this method. There is no limit to the number of tables which can be joined.

5.2 Database Generation

We generate a hypothetical database with the following tables:

- Persons Table: ID, FirstName, MiddleName, LastName, Gender, Dependents, Age.
- Dependents Table: ID, CareTakerID, DependentID.
- Companies Table: ID, Name, CompanyType.
- Employees Table: ID, PersonID, CompanyID, EmployeeType, Salary.
- Customers Table: ID, PersonID, CompanyID, CustomerType.
- Addresses Table: ID, PersonID, City, Street, Nation, Zipcode, IsPrimaryAddress.
- Phones Table: ID, PersonID, PhoneNumber, IsPrimaryPhone.

After the database is generated, a cell is created for each datum and inserted into a single ant. Using an interleaved allocation scheme, these ants are distributed across all nodes currently connected to the program's network.

Simple and complex queries are generated to access information in these tables. Complex queries join multiple tables. For instance, the query asking the question "What

First Iteration	Fifth Iteration
871.8 ms	371.8 ms

Table 1. Average query completion time. An iteration is the execution of a single query.

types of companies hire people less than 18?" joins the Persons, Companies and Employees tables, and is written as follows:

> SELECT CompanyType FROM Companies WHERE ID EQUAL SELECT CompanyID FROM Employees WHERE PersonID EQUAL SELECT ID FROM Persons WHERE Age LESS '18'.

6 Experiments and Results

A complete set of experiments should investigate the performance of the load-balancing algorithm in the following areas: *i)* dynamic clustering of similar data; *ii)* monitoring system changes with addition and deletion of nodes; *iii)* replicating data as the needs of the system vary; and *iv)* reducing the time required to access data as the learning process adapts to changing requirement.

We present here only the preliminary experiments, designed to investigate the algorithm's ability to:

1. Reduce the time required to complete a set of queries.
2. Group, within a single ant, cells accessed with similar frequency by a single node.

We created a database of seven tables. The seven tables decomposed to 226 cells of data; the program created an ant for each cell. The ants, each containing a single cell, were distributed to five nodes (workstations) using the interleaved allocation method of dispersal. We composed ten queries for this database; two different queries were assigned to each of the five nodes. Each query was repeated five times so that a total of 50 queries were performed, with ten from each node. The queries were submitted, in randomized order, every 30 seconds.

The results indicate the algorithm's potential to dynamically re-distribute, and, if necessary, partially replicate, the load of information on each node, thus reducing query completion times. The relatively long completion time for the first iteration (Table 1) is due to the fact that some of the required cells were located on remote nodes; numerous lengthy communications were required to complete even a simple query. After five iterations of each query and about 35 iterations of the load balancing algorithm on each node, ants containing these cells had joined and split appropriately. They then sent replicas to nodes needing the information, thus reducing completion times by about one third.

	Start	Finish
Total number of ants	226	171
Total number of cells	226	266
Number of single-celled ants	226	160
Number of multi-celled ants	0	11
Average number of cells per multi-celled ant	0	9.6

Table 2. Grouping information at the beginning and the end of the experiment. A single-celled ant contains only one cell; multi-celled ants contain more than one cell.

As the experiment progressed, cells accessed at a similar frequency by a particular node were grouped together in a single ant. One indicator of this grouping behavior is the number of cells contained within an ant (Table 2). At the beginning of the experiment, all ants are single-celled. During the experiment, the total number of ants decreased from 226 to 171, whereas the total number of cells increased from 226 to 266. At the conclusion of the experiment, all cells of data needed to complete the ten different queries were located in a total of 11 ants spread across all five nodes, with an average of 9.6 cells per multi-celled ant. All 160 single-celled ants contained a cell never required by any of the queries. The increase in the number of cells indicates that partial replication of data occurred.

The algorithm groups similar cells into a few ants. Similarity is defined as a similar rate of access by a single node, rather than a relationship within a database table, or between database tables. Notably, all 106 cells needed to complete the ten queries were grouped in only 11 out of a total of 171 ants. As grouping progressed, the number of ants decreased despite replication of cells required by more than one node. One of the nodes completed two simple queries which accessed cells in two different tables. At the experiment's conclusion, all ten cells required to complete these two diverse queries resided in only one ant on this node. This grouping capacity facilitates migration of similar cells.

7 Conclusions and Future Work

We have designed an adaptive swarm-type system of mobile agents, called Pogo ants, to manage a partially replicated database. The system performs dynamic, event-driven, completely distributed load balancing with partial replication. Every node of the network is capable of producing transactions which trigger events in the network. Load balancing runs with lower priority, in the background, and

the learning process is not synchronized with the events. The methodology was illustrated by an example in which only query transactions were performed.

Due to the short-term memory of the connection between cells and the nodes requesting their information, the algorithm has the ability to dynamically adjust to changing requirements. Further experiments to demonstrate this dynamic load balancing capacity are currently in progress.

Distributed data replication has applications not only in distributed databases [26], but also in fault-tolerant shared memory clusters [28], P2P file sharing services [27], data migration in dynamic content Web servers [33], replication for Web hosting systems [32], mobile environments [7] and Web distribution of XML documents [2]. There is recent interest in adaptive replication procedures for Web services.

The system ensures a minimum number of replicas; the maximum number of replicas is the number of nodes in the network. The replicas have a short lifespan. They continue to exist only to the extent that they are periodically fed by local events. In this manner, unneeded replicas are eliminated from the system and the number of replicas converges to an optimum, determined by the event patterns.

In our model, replicas of data objects are created dynamically on nodes which frequently access these objects. This replication scheme is more complex than simple caching, since copies can also be deleted. Access frequency is a simple criterion and we may consider replicating data objects based on their relevance for information retrieval, as in [21].

Replicated databases must synchronize data replicas after each update [26]. An appropriate technique would be to let data be accessed without a priori synchronization, based on the optimistic assumption that problems will occur only rarely, if at all [30]. As in the replication mechanism, updates could be propagated in the background, by swarms of specialized Pogo ants. In our system, data updates and queries trigger events in the system. Each event results in a dynamic load-balancing process which runs in the background. This approach faces the challenge of diverging replicas and conflicts between concurrent operations. It is thus applicable only for applications that can tolerate occasional conflicts and inconsistent data.

An interesting result reported in [22] is that a swarm of heterogeneous ants (ants with diverse parameters) can perform better than a homogeneous swarm. Moreover, dynamically switching the behavioral pattern of the agents may also improve the performance of the clustering algorithm. For instance, in [31], agents have the autonomy to define their own migration policy as well as the migration of other agents. We intend to investigate these aspects in the future. We also intend to track the load-balancing process using measures, like the ones reported in [22].

References

[1] S. Abdul-Wahid, R. Andonie, J. Lemley, J. Schwing, and J. Widger. Event-driven load balancing of partially replicated objects through a swarm of mobile agents. In B. Kovalerchuk, editor, *Proceedings of the IASTED International Conference on Computational Intelligence (CI 2006)*, pages 110–115. ACTA Press, 2006.

[2] S. Abiteboul, A. Bonifati, G. Cobéna, I. Manolescu, and T. Milo. Dynamic XML documents with distribution and replication. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 527–538. ACM Press, 2003.

[3] A. Abraham and V. Ramos. Web usage mining using artificial ant colony clustering and genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1384–1391. IEEE Press, 2003.

[4] K. P. Adams, D. Graanin, and M. G. Hinchey. Increasing resiliency through priority scheduling of ssynchronous data replication. In *Proceedings of the 11th International Conference on Parallel and Distributed Systems*, pages 356–362. IEEE Computer Society, 2005.

[5] P. Apers. Data allocation in distributed DBMS. *ACM Transactions on Database Systems*, 13(3):263–304, 1988.

[6] M. Bedal, J. Gaber, H. El-Sayed, and A. Almojel. Swarm intelligence. In S. Olariu and A. Y. Zomaya, editors, *Handbook of Bioinspired Algorithms*, pages 55–84. Chapman & Hall/CRC, 2006.

[7] A. Beloued, J.-M. Gilliot, M.-T. Segarra, and F. André. Dynamic data replication and consistency in mobile environments. In *Proceedings of the 2nd international doctoral symposium on Middleware*, pages 1–5. ACM Press, 2005.

[8] E. Bonabeau, M. Dorigo, and 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.

[9] E. Bonabeau, A. Soblowski, G. Theraulaz, and J. L. Deneubourg. Adaptive task allocation inspired by a model of division of labour in social insects. Technical report, Santa Fe Institute Working Paper 98-01-004, 1998.

[10] A. R. Chaturvedi, C. A. K., and R. J. Scheduling the allocation of data fragments in a distributed database environment: a machine learning approach. *IEEE Transactions on Engineering Management*, 41(2):194–207, 1994.

[11] J. L. Deneubourg, S. Gross, N. R. Franks, A. Sendova-Franks, C. Detrain, and L. Chrétien. The dynamics of collective sorting: robot-like ants and ant-like robots. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 356–363. MIT Press, 1991.

[12] D. M. Gordon. *Ants at work: how an insect society is organized*. Free Press, New York, 1999.

[13] D. M. Gordon. The organization of work in social insects colonies. *Complexity*, 8(1):43–46, 2003.

[14] J. Handl and B. Meyer. *Improved ant-based clustering and sorting in a document retrieval interface*, pages 913–923. Springer-Verlag, 2002.

[15] V. Hartmann. Evolving agent swarms for clustering and sorting. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 217–224. ACM, 2005.

[16] J. Holliday, D. Agrawal, and A. E. Abbadi. Partial database replication using epidemic communication. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 485–493. IEEE Computer Society, 2002.

[17] D. Kossmann. The state of the art in distributed query processing. *ACM Computing Survey*, 32(4):422–469, 2000.

[18] T. Loukopoulo and I. Ahmad. Static and adaptive distributed data replication using genetic algorithms. *Journal of Parallel and Distributed Computing*, 64(11):1270–1285, 2004.

[19] T. Loukopoulos and I. Ahmad. Static and adaptive data replication algorithms for fast information access in large distributed systems. In *Proceedings of the 20th International Conference on Distributed Computing Systems*, pages 385–392. IEEE Computer Society, 2000.

[20] K. H. Low, W. K. Leow, and M. H. Ang, Jr. Task allocation via self-organizing swarm coalitions in distributed mobile sensor network. In *Proceedings of the 19th National Conference on Artificial Intelligence*, pages 28–33, 2004.

[21] Z. Lu and K. McKinley. Partial replica selection based on relevance for information retrieval. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 97–104. ACM Press, 1999.

[22] E. D. Lumer and B. Faieta. Diversity and adaptation in populations of clustering ants. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animals*, pages 501–508. MIT Press, 1994.

[23] A. Montresor, H. Meling, and O. Babaoğlu. *Messor: Load-Balancing through a Swarm of Autonomous Agents*, pages 125–137. Springer-Verlag, 2003.

[24] R. Mukkamala, S. C. Bruell, and R. K. Shultz. Design of partially replicated distributed database systems: an integrated methodology. In *Proceedings of the 1988 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 187–196. ACM Press, 1988.

[25] T. Nakano and T. Suda. Self-organizing network services with evolutionary adaptation. *IEEE Transactions on Neural Networks*, 16(5):1269–1278, 2005.

[26] M. Nicola and M. Jarke. Performance modeling of distributed and replicated databases. *IEEE Transactions on Knowledge and Data Engineering*, 12(4):645–672, 2000.

[27] A. Oram. *Peer-to-peer: Harnessing the power of disruptive technologies*. O'Reilly, 2001.

[28] M. D. Peysakhov and W. C. Regli. Dynamic data replication: an approach to providing fault-tolerant shared memory clusters. In *Proceedings of the Ninth Annual Symposium on High Performance Computer Architecture*, pages 203–214, 2003.

[29] M. D. Peysakhov and W. C. Regli. Ant inspired server population management in a service based computing environment. In *Proceedings of the IEEE Swarm Intelligence Symposium, Pasadena, California*, pages 357–364, 2005.

[30] Y. Saito and M. Shapiro. Optimistic replication. *ACM Computing Surveys*, 37(1):42–81, 2005.

[31] I. Satoh. Bio-inspired deployment of distributed applications. In *Proceedings of the 7th Pacific Rim International Workshop on Multi-Agents*, pages 243–258. Springer-Verlag, 2004.

[32] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. van Steen. Replication for web hosting systems. *ACM Compututing Surveys*, 36(3):291–334, 2004.

[33] G. Soundararajan and 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*, pages 268–282. IBM Press, 2005.

[34] J. Suzuki and T. Suda. A middleware platform for a biologically inspired network architecture supporting autonomous and adaptive applications. *IEEE Journal on Selected Areas in Communications*, 23(2):249–260, 2005.

[35] A. L. Vizine, L. N. de Castro, E. R. Hruschka, and R. R. Gudwin. Towards improving clustering ants: an adaptive ant clustering algorithm. *Informatica*, 29(2):143–154, 2005.

[36] M. Wang and T. Suda. The bio-networking architecture: A biologically inspired approach to the design of scalable, adaptive, and survivable/available network applications. In *Proceedings of the Symposium on Applications and the Internet*, pages 43–56. IEEE Computer Society, 2001.

[37] O. Wolfson, S. Jajodia, and Y. Huang. An adaptive data replication algorithm. *ACM Trans. Database Syst.*, 22(2):255–314, 1997.

[38] O. Wolfson and A. Milo. The multicast policy and its relationship to replicated data placement. *ACM Trans. Database Syst.*, 16(1):181–205, 1991.