CHAPTER 6

Bioinspired Computing: Swarm Intelligence

Brains exist because the distribution of resources necessary for survival and the hazards that threaten survival vary in space and time.

—John M. Allman, Evolving Brains

Natural systems solve multifaceted problems using simple rules, and exhibit organized, complex, and intelligent behavior. Natural process control systems are adaptive, evolutionary, distributed (decentralized), reactive, and aware of their environment. *Bioinspired computing* (or *biologically inspired computing*) is a field of study that draws its inspiration from the sophistication of the natural world in adapting to environmental changes through self-management, self-organization, and self-learning. Bioinspired computational methods produce informatics tools that are predicated on the profound conceptions of self-adaptive distributed architectures seen in natural systems. Heuristics that imitate these natural processes can be expressed as theoretical methods of constrained optimization. Such heuristics define a *representation*, in the form of a fitness function. This function describes the problem, evaluates the quality of its solution, and uses its *operators* (such as crossover, mutation, and splicing) to generate a new set of solutions.

Ashby's (1952) book *Design for a Brain* discusses the mechanisms that shape the concept of adaptive behavior, as demonstrated in living organisms, and the adaptive behavior of the brain. The author defines *adaptation* as a form of behavior that promotes stability and that maintains the essential variables, within physiological limits. Additionally, stability is expressed as a combined function of multiple fields with changing dynamics. Therefore, stability is assumed to be associated with a coordination function between various fields. As the system and feedbacks become more complex, the achievement of stability becomes more difficult, and the likelihood of instability, greater.

Biologically, an important factor in the survival of an organism is its ability to maintain its essential variables, within viable bounds. Otherwise, the organism faces the possibility of disintegration or loss of identity (dissolution, death), or both. Adaptation provides an organismic stability criterion that contributes to the maintenance of the essential variables, within viable limits; an adaptive system is a stable system (Harvey et al. 2005, the region of stability being that part of the state space where all essential variables are within physiological limits.

In the natural world the brain exhibits the properties of a highly efficient informatics tool that gathers data (sensor function), infers and stores useful patterns in the data (knowledge base, memory), uses that data for planning and anticipating future actions (decision making), executes those actions (control functions), and learns from the consequences of those actions (learning). The brain acts as an information processing machine that enables a fast and adequate response to environmental perturbations by filtering disrupting triggers.

Jacob, Lanyon-Hogg, Nadgir, and Yassin (2004) conceptualized autonomic computing as analogous to the *autonomic nervous system* (ANS), which constitutes an essential element of the peripheral nervous system. Autonomic computing resembles the ANS, insofar as the latter is composed of a hierarchy of numerous self-governing components that give monitoring and control functions the ability to transmit messages to various organs at a subconscious level. While the ANS monitors the "operating environment," it also maintains the required equilibrium by enacting the optimal changes at a subconscious level. In general, the ANS is responsible for controlling various actions related to digestion, perspiration, heart rate, respiration, salivation, pupil dilation, and other such functions. The ANS facilitates such control systems by actively monitoring, integrating, and analyzing input stimuli via sensory channels and distributing electrochemical impulses via motor channels to generate control responses to various environmental conditions.

The brain and the ANS inspire us with design principles abstracted to informatics tools, such as *artificial neural networks* (ANNs) and *autonomic computing*. But, nature motivates us with many more naturally occurring and highly efficient computational phenomena that, when modeled effectively, can improve the use of computers in solving complex optimization problems. Such phenomena exist in the form of social interactions, evolution, natural selection, biodegradation, swarm behavior, immune systems, cross-membrane molecular interactions, and so on. Software- or field programmable gate array– (FPGA-) based agents can model these natural forms of computational and collective intelligence as evolutionary algorithms, swarm intelligence (SI), artificial immune systems, artificial life, membrane computing, DNA computing, quantum computing, and so on. The abstractions derived from natural processes formalize the distributed computing paradigm, in which independent entities improve their reactive behaviors by interacting with other entities, using a well-defined protocol, and fine-tuning their control actions.

Applications

Bioinspired computing systems confront complex problems by exploiting the design principals and computational techniques encountered in natural processes. These systems possess a deep understanding of the distributed processes that exist in nature and use the concepts of theoretical biology to produce informatics tools that are robust, scalable, and flexible.

Evolvable Hardware

Evolvable hardware (EHW) is a novel field, in which practical circuits that exhibit desirable behaviors are synthesized, using evolutionary algorithms. In their rudimentary configuration, such algorithms—such as *genetic algorithms* (GAs)—influence a population of existing circuits to synthesize a set of new candidate circuits targeted to fulfill the design specifications. The quality of the circuit is evaluated, using a fitness function that ascertains if all the design requirements are met. Such techniques are useful when design specifications merely specify the desired behavior or when hardware needs to adapt dynamically and autonomously to changing operating conditions. In both cases, design specifications lack adequate information to warrant the use of conventional methods. An evolvable circuit can be synthesized, using a simulation tool (such as SPICE), a physical device (such as an FPGA), or a configurable logic.

In an evolutionary design approach it is not necessary to have a priori knowledge of the problem domain. In many cases, it may be either too complex or too expensive to acquire such information. As the complexity of the circuitsgrows, it becomes increasingly challenging to comprehend the dynamics between the various components of the circuit. EHW envisages evolutionary design techniques that facilitate development of online hardware that adapts its architecture, according to environmental changes or perturbations.

An example of this technique is *cache quality of service* (CQoS) logic (Iyer 2004), which performs dynamic partitioning of the cache among selected cores to improve the performance of the system. The automation methodology employs central processing unit (CPU) performance counter feedback in a GA to evolve an optimal cache distribution scheme. The GA chromosome contains all the building blocks for a solution to the problem at hand in a form that is suitable for the genetic operators and the fitness function.

Each CPU core node is represented by an *n*-bit binary number, called a *gene*. These *n*-bit genes define the representation, in which each bit and its position correspond to an individual cache slice slot of the total *last-level cache* (LLC). The GA-based evolutionary algorithm dynamically partitions the cache into private and shared regions, without prior knowledge of the workload profile, and allows sharing among the cores. The advantages of evolutionary methodology in partitioning the cache are that it is practical and significantly reduces the overall miss rate of the cache, at the cost of a small evaluation (training) time overhead. This methodology ensures optimal cache partitioning, with a view to increasing the *instructions per cycle* (IPC) and reducing the cache miss rate, thus ultimately enhancing overall performance. Each slot is assumed to host the same-sized cache partition. Figure 6-1 depicts a chromosome structure with an 80-bit string (20 bits per core) that represents the association of each core with a 1MB cache slot.

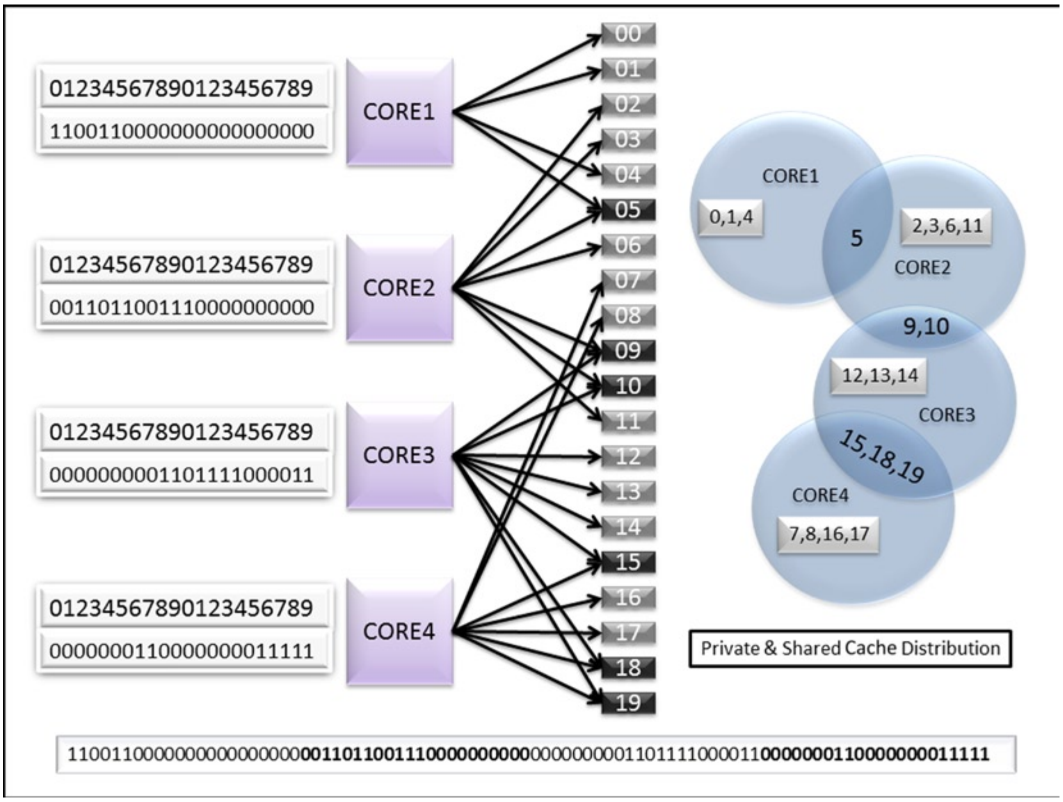

Figure 6-1. *The chromosome structure of the cache clustering, based on workload behavior, using four cores and 20 cache slots sized 1MB. For example, for core 1 cache slots, 0, 1, and 4 are private, and cache slot 5 is shared with core 2. The complete string describes the cache association of each core with a cache slot*

The cache-clustering fitness function is a weighted function capable of measuring the quality or performance of a solution—in this case, a cache-partitioning scheme that improves workload performance among CPU cores. This function is maximized by the GA system in the process of evolutionary optimization. A fitness function must include and correctly represent all or at least the most important factors that affect the performance of the system. You may need functions that represent workload-independent and workload-dependent characteristics. The workload-independent function in Equation 6-1 executes the global fair allocation of cache, without biasing the characteristic behavior of a workload. This function, where N is the number of workloads, and (ϕ_k) is the miss rate of each workload (see Equation 6-2), attempts to maximize the average performance (F_{avg}) of all workloads. The contribution of each workload k can be weighted (λ_k), based on the effect of throughput by individual workload.

$$F1_{avg} = 1.0 - \sum_{k=1}^{N} \lambda_k \cdot \phi_k, \ \sum_{k=1}^{N} \lambda_k = 1.0 \tag{6-1}$$

$$\phi_k = \frac{LLC_k^{miss}}{LLC_k^{miss} + LLC_k^{hit}} \tag{6-2}$$

The workload-dependent function (see Equation 6-3) biases the characteristic behavior of a workload and tries to improve performance, based on that characteristic. This function, where (ϕ_k) is the fraction of the miss rate of each workload over the total miss rate of the cores, and (ψ_k) is the fraction of the cache size allocated to each core over the total cache size, allows us to identify certain characteristics that may be capable of boosting the service-level objectives for a given environment. As shown in Equation 6-3, a certain bias can be generated to build pressure for allocation of cache sizes (S_k) for each core k that is proportional to the LLC miss rate ratio of that core (see Equation 6-4).

$$F2_{avg} = 1.0 - \sum_{k=1}^{N} \lambda_k \cdot abs\left(\frac{\varphi_k}{\psi_k} - 1.0\right) \tag{6-3}$$

$$\varphi_k = \frac{\phi_k}{\sum_{j=1}^{N} \phi_j} \tag{6-4a}$$

$$\psi_k = \frac{S_k}{\sum_{j=1}^{N} S_j} \tag{6-4b}$$

Finally, the overall fitness (see Equation 6-5) can be defined as the weighted proportion of each individual fitness, as given in Equations 6-1 and 6-3:

$$F_{avg} = \alpha \cdot F1_{avg} + (1-\alpha) \cdot F2_{avg}. \tag{6-5}$$

Bioinspired Networking

Communication and network technologies have gained a lot of traction in recent years, owing to advancements in *cloud-based networking* (CBN), networked embedded systems, *wireless sensor networks* (WSNs), the *Internet of Things* (IOT), *software-defined networking* (SDN), and so on. Furthermore, enterprise-class networking solutions are being developed to deliver high resiliency, high availability, and high reliability through capacity planning, traffic engineering, throughput management, and overlaying of multitenant applications, using the existing Internet infrastructure. As the network scales, the search space for the optimal route increases dramatically. The number of routing tables and the amount of traffic overhead overwhelm network bandwidth. Ideally, we would like to have efficient, self-organizing networks with low route-finding latencies (or overhead) and high probabilities of successful transmission.

Nevertheless, significant challenges prevent us from realizing the practical implementation of new networking paradigms. In addition to the need for scalability, availability, and survivability, these challenges arise from resource constraints, absence of centralized architecture, and the dynamic nature of networking. However, similar phenomena are also found in natural processes that are successfully dealt with through adaptation in biological systems. Biological communication paradigms are evolutionary; resilient to failure; adaptive to environmental conditions; and collaborative, on the basis of a simple sets of rules.

Bioinspired networks self-organize by apprehending the mutual interactions of components that produce complex patterns. These interactions facilitate behavioral responses through information transfer between interacting components as well as through the interaction of components with the environment.

Gianni, Ducatelle, and Gambardella (2005) presented the AntHocNet design for stigmergy-driven shortest-path discovery, based on a self-organizing behavior exemplified in ant colonies. Similar routing algorithms exist for packet-switched networks, such as AntNet (Di Caro and Dorigo 1998), and circuit-switched networks, such as artificial bee colony (ABC) (Schoonderwoerd et al. 1997). The AntHocNet algorithm inserts limited routing information in "Hello" messages so that the information regarding existing paths can propagate throughout the network, using node-to-node information exchange. This process is equivalent to collective ant learning behavior, in which ants swarm together to gather and maintain updated information. Artificial ants instigate the stigmergic communication process by acting as autonomous agents that update and follow the pheromone table (path). Similar to routing ants, the pheromone table explores the high-probability paths that can be used for routing data packets. Additionally, ants put their limited resources toward optimizing global behavior to identify the food source in a cost-effective manner. This behavior inspires resource-efficient networking techniques.

Given their dynamic nature and lack of infrastructure, networks are also prone to failure and delay. Therefore, networks should have capability to self-organize and self-heal in real time. Dynamic networks (especially mobile ad hoc networks) may use the bioequivalent of epidemic models both to describe and to adapt information dissemination. Papadopouli and Schulzrinne (2000) described a simple stochastic epidemic model that estimates the delay until data diffuse to all mobile devices. Carreras et al. (2006) proposed an *epidemic spreading mechanism* for efficient information dissemination in clustered networks and for opportunistic routing in delay-tolerant networks. The authors used *eigenvector centrality* (EVC) as an objective fitness measure of the ability of nodes to spread an epidemic (information) within the network. The resulting topology built using EVC defines the regions in which the epidemic spreads extremely fast. Infection fronts (information) spread toward highly connected neighborhoods (EVC), because spreading is fastest there.

Resource-constrained sensors, such as wireless sensors, are limited in energy, bandwidth, storage, and processing capabilities. Large numbers of such sensors create a sensor management problem. At the network layer the solution entails setting up an energy-efficient route that transmits the nonredundant data from the source to the sink to maximize the battery's and sensors' lives. This is done while adapting to changing connectivity resulting from the failure of some nodes and the powering up of new nodes. Khanna, Liu, and Chen (2009) demonstrated that the GA-based approach optimizes the sensor network to maximize energy usage as well as battery conservation and route optimization. Each sensor is encoded with a gene that identifies it and any other specific information it may contain. This information may be related to sensor objectivity, next hop, cluster domain, and so on. The GA adaptation process evolves optimal cluster boundaries, in terms of addition, deletion, or modified sensor objectives. The process also discovers optimal routes from cluster heads to the sink.

Datacenter Optimization

The size and complexity of modern distributed datacenter systems are expanding every day. The volume of information that must be processed in real time has been growing geometrically over the past few years, requiring peak processing capabilities to rise in concert. Despite the superior performance per watt that newer platforms deliver, handling peak loads continues to call for higher power delivery and heat dissipation capacities per cubic meter in enterprise information technology (IT) and datacenter facilities, with 63 percent of the total cost of ownership going toward powering, cooling, and electricity delivery infrastructure. In contrast to the traditional focus on delivering the highest throughput or lowest response time, unconstrained by power, these realities have made it a more compelling proposition to minimize the amount of energy consumed, relative to computational work performed, while meeting responsiveness targets. In particular, dynamically conserving power when some machines do not need to be in full use translates directly into cost savings and creates greater allowance for other, more power-constrained servers.

One way to optimize power in a datacenter is to regulate or redistribute the load of a rack-level server unit autonomously through management of job admission, distribution, and continuous balancing. Barbagallo et al. (2010) put forward self-organizing architectures for dynamic workload distribution, using a decentralized approach built on top of *SelfLet architecture*. SelfLet architecture (Devescovi et al. 2007) is a bioinspired system that possesses the capability to change and adapt its internal behavior dynamically, according to variations in the environment. SelfLet uses autonomic reasoning to facilitate self-management capabilities. SelfLet itself represents a service framework built using a self-sufficient piece of code that interacts with a group of other SelfLet individuals and that cooperates through high-level functions. Each SelfLet either offers or consumes a service and interacts via a communication framework. The authors used self-organization algorithms based on the principles of collective decision making in animal colonies. Self-organizing in these colonies is characterized by scouting, evaluating, deliberating, and decision-making functions. Self-organization in a datacenter can be summarized using similar entities, as follows:

- *Colony*: A collection of *virtual machines* (VMs) residing on the same server.
- *Scout*: Explores multiple physical servers and compares them with the original one. A scout can be characterized by its current location, lifetime, and information stored related to each server class that it examined.
- *Server manager*: Communicates with the scouts and makes decisions related to the movements of VMs.

Based on the collective data from multiple scouts, a decision is made either to permit or to inhibit migration. As in biological systems, the decision whether to migrate is not deterministic and follows a probability distribution. This avoids a reactive migration, which could result in instability and oscillatory behavior. Furthermore, like biological systems, individual servers may propagate an inhibitor flag to prevent migration in the middle of critical operations.

Li and Parashar (2003) developed *AutoMate architecture* to investigate bioinspired conceptual models and implementation designs for developing and executing self-managing (i.e., configuring, healing, optimizing, and protecting) grid applications, while dealing with the challenges of complexity, dynamism, and heterogeneity. AutoMate architecture is built on three operative principles:

- Separating policies from the mechanisms related to algorithms, communication protocols, and so on that drive those policies
- Applying context-, constraint-, and aspect-based composition techniques to applications to synthesize dynamic requirements for compute resources, performance guarantees, and QoS
- Developing proactive- and reactive-component management to optimize resource utilization and application performance in dynamic environments

Bioinspired Computing Algorithms

Bioinspired computing methods are metaheuristics that imitate methods for solving optimization problems in natural processes. These heuristics deliver a robust and decentralized compute engine that can perform in noisy ecosystems and yet deliver a desired behavior, while operating within time, energy, and power constraints. Such computing methods have been used in almost all areas of optimization, knowledge discovery, and big data analytics, including computer networks, image processing, WSNs, security, control systems, biomedical systems, and robotics. The following sections give a brief overview of bioinspired optimization algorithms that are computationally efficient alternatives to the traditional deterministic approach—an approach that does not scale well and that requires massive computational effort. Designing bioinspired algorithms involves identifying a suitable representation of the problem, developing a fitness function to evaluate the quality of solution, and defining operators to produce a set of new solutions. Broadly, bioinspired algorithms can be divided into three major classes: *evolutionary* algorithms, *swarm-based* algorithms, and *ecological* algorithms. These classes are further divided into subclasses, based on their inspiration from specific cases of naturally occurring processes involving ants, fireflies, bacteria, bees, birds, and so on.

Swarm Intelligence

Swarm intelligence (SI) is a type of artificial intelligence based on the collective behavior of decentralized, self-organized systems. The term was introduced by Beni and Wang (1989), in the context of cellular robotic systems. SI typically comprises a collection of unsophisticated agents, called *boids*, that interact locally with other agents as well as with the environment, using extremely elementary rules. No centralized control infrastructure governs an individual agent's behavior or interactions. Instead, local and random interaction between participating agents leads to an intelligent global behavior unattributable to individual agents. In other words, collective interaction at peer level leads to a sophisticated phenomenon globally. Natural examples of SI include ant colonies, bird flocking, animal herding, bacterial growth, and fish schooling. SI-inspired systems include positive feedback, negative feedback, amplification of fluctuations, and multiple interactions between multiagents.

Autonomic computing and SI are closely related. For example, through the tuning of system parameters, the self-configuration aspect can be achieved autonomically rather than manually. In the natural world the local process indicators can change, be reinforced, or reach a threshold, reflecting the actual dynamic swarm situation. Clearly, system performance can be optimized through interaction between multiple local agents. Also, system robustness can be guaranteed through this kind of parallel multiagent interaction. Finally, security and load balancing can be fulfilled with careful parameter and rule design. The appropriate tradeoff between the purely reactive behavior promoted by traditional stigmergy and the purely cognitive behavior promoted by artificial intelligence approaches has to be determined. Stigmergy is a mechanism occurring in many social animal societies that contrives to solve complex problems using a decentralized approach in a self-organized system. This system rewards positive feedback, while penalizing the negative. As a result, the system enables a complex, intelligent infrastructure that needs no planning, control, or complex interactions between the agents. The social aspects support efficient collaboration between elementary agents, which lack memory, intelligence, and even awareness of each other.

Ant Colony Optimization Algorithm

In the natural world, ants wander randomly until they find a path that leads to food. This behavior has inspired a variant of SI called the *ant colony optimization* (ACO) algorithm. In the ACO scenario (Dorigo, Di Caro, and Gambardella 1999) ants communicate with other ants to exchange status information regarding food sources through changes in the environmental medium by depositing pheromones. The status information is exchanged within a local scope, and the context is transferred only to ants at the location of pheromone deposition. After finding food, these ants return to the colony, while laying down pheromones for other ants to follow. Upon discovering the food trail, the other ants abandon their random food search and follow the pheromone trail, thereby reinforcing the original path.

Over time, however, the pheromone trail tends to evaporate and lose its attractive property. If the round-trip time between the ant colony and the food source is great, the pheromone can evaporate faster than it can be reinforced. In contrast, a short pheromone trail has a greater pheromone density and can be reinforced faster than it evaporates. The pheromone evaporation process is analogous to avoiding convergence to a local optimal; the process avoids strengthening the nonoptimal solution, while aiding the unconstrained exploration of the solution space. In general, variation in the pheromone quantity on the edge allows for the choice of a specific edge. A stable system is a network of strong edges that adapt to active variations and environmental changes. Because of dynamic variations in the interactive environment, certain edges are reinforced through positive feedback, whereas others are weakened because of negative feedback. A slight variation in the edges can result in an alternate route that remains valid until other edges exhibit stronger traits. For instance, if the pheromone trail to a single food source is reinforced, it will take much longer to discover alternate sources that may be optimally suited for the colony, in terms of distance or abundance. ACO attempts to find a path to the solution that is likely to be followed by other agents, thereby building positive feedback that eventually leads to a single path to the solution.

ACO methodologies differ from evolutionary algorithms in one main regard. In evolutionary algorithms, such as GA, all knowledge about the problem is contained in the current population, whereas in ACO algorithms a memory of past performance is maintained in the form of pheromone trails Dorigo, Di Caro, and Gambardella (1999) represented ants as playing the role of environmental signals, and the pheromone update rule, the role of the automaton learning rule. In ACO the environmental signals/ants are stochastically biased, by means of their probabilistic transition rule, to direct the learning process toward the most interesting regions of the search space. That is, the whole environment has a key, active role in the learning of good state-action pairs. The basic ACO rule can be defined by the following process (Engelbrecht 2006; Wang et al. 2007):

1. Create *nr* global ants; each ant visits each food source exactly once.
2. Evaluate the fitness of each food source; a distant food source has a lesser probability of being chosen.
3. Update the ants' pheromone and the age of weak regions.
4. Move local ants to better regions, based on the pheromone intensity, to improve their fitness; otherwise, choose new random search directions.
5. Update the ants' pheromone to all the regions they traversed.
6. After each iteration, evaporate the ants' pheromone.

Ants are attracted to the regions, based on the intensity of the pheromone at time *t*. As the pheromone evaporates, that region becomes less attractive to the ant and is finally abandoned. The probability of an ant *k* transitioning from region *i* to region *j* at time *t* is

$$P_k(i,j,t) = \frac{\tau_{i,j}(t)^{\alpha} \cdot d_{i,j}(t)^{\beta}}{\sum_{n \in N_i^k} \tau_{i,n}(t)^{\alpha} \cdot d_{i,n}(t)^{\beta}}, \text{ if } \ j \in N_i^k, \tag{6-6}$$

where

$\tau_{i,j}(t)$ = pheromone trail region, represented by edges *i* and *j* at time *t*

$d_{i,j}(t)$ = distance between source (*i*) and destination (*j*) locations

N_i^k = feasible neighborhood of ant *k* at source *i*

α = relative significance of the pheromone trail

β = relative significance of the distance between source and destination

The positive parameters α and β define the relationship between pheromone and heuristic information. Therefore, the probability of a trail that is chosen by ant *k* is a function of distance and the density of the pheromone that already exists on that trail at time *t*. The significance of intensity and distance to the cost function is determined by α and β, respectively. Thus, the better the region is, the more attraction it has to the successive ants. The pheromone concentration in the region is updated as a function of constant evaporation (*p*) and new deposits ($\Delta\tau_k$) by the ants attracted to this region, such that

$$\tau_{i,j}(t+1) = p \cdot \tau_{i,j}(t) + \sum_{k=1}^{m} \Delta\tau_{k,i,j}(t) \tag{6-7}$$

$$if \ \ (i,j) \in S_k : \Delta\tau_{k,i,j} = \frac{1}{L_{k,i,j}}, \tag{6-8}$$

where

p = evaporation constant, whose value can be set between 0 and 1 and represents the rate at which the pheromone evaporates

$L_{k,i,j}$ = length of the tour by ant *k*, with shorter tours resulting in higher pheromone density

$\Delta\tau_{k,i,j}$ = amount of pheromone deposited by ant *k* in region (*i, j*).

The probability of local ants' selecting a region is proportional to its pheromone trail. The pheromone is affected by the evaporation rate, ant age, and growth of fitness. Thus, this pheromone-based selection mechanism is capable of promoting the solution candidate update, which is suitable for handling the changing environments in optimization.

ACO techniques can be used for knowledge discovery corresponding to learning the functional relationship between variables, changes in data pattern, and data classification. The social behavior of the ants suggests the notion of formulating an infrastructure that fosters the concept of self-organization, using natural interactions and local information to solve complex computational problems.

Dorigo, Di Caro, and Gambardella (1999) described a solution to the Traveling Salesman Problem (TSP), in which *m* artificial ants concurrently build a tour of the TSP. Initially, *k* ants are placed on randomly selected cities. At each step the *k*th ant applies a probabilistic action choice rule to resolve which city to visit next. The probability that an ant chooses to travel from city *i* to city *j* ($J \in N_i^k$) is given in Equation 6-6, where N_i^k defines a potential neighborhood of *k* cities when the ant is in city *j*. If $\alpha = 0$, the closest cities are more likely to be selected; if $\beta = 0$, then only pheromone amplification is used, without any heuristic bias. Each ant *k* retains a memory area, where it generates the tour, computes the length of the tour, and retraces the path to deposit the pheromone to the arcs of the tour. Pheromone trails are updated after all ants have constructed their tour. This is done by decreasing the pheromone by a constant factor on all arcs (evaporation) and then successively adding the pheromone to the arcs crossed by ants in their tours. Pheromone evaporation and updating are implemented in Equation 6-7. Arcs that are part of a short route and are visited by many ants receive more pheromone and are therefore more likely to be chosen by ants in future iterations of the algorithm.

Particle Swarm Optimization

Particle swarm optimization (PSO) is a stochastic computational technique that iteratively optimizes the candidate solution of a problem until it attains the target fitness (or quality) (Kennedy and Eberhart 1995). This technique is biologically inspired by the social behavior of bird flocking and fish schooling. Owing to its simplicity and computational efficiency, PSO has been successfully applied to many engineering research and optimization applications. *Particle* in PSO denotes an individual member of a population that searches for optimal behavior when subjected to velocity and acceleration in a large search space. Each particle in the swarm explores the coordinates of the solution space and records the following four vectors, relative to the best solutions (fitness) achieved in that process:

- Particle's current coordinates
- Particle's velocity, with respect to magnitude and direction
- Coordinates (position) associated with the particle's local best solution achieved up to that point (*pbest*)
- Coordinates (position) associated with the particle neighborhood's best solution achieved up to that point (*gbest*)

PSO can search a large solution space, while making no assumptions regarding the problem being optimized. PSO looks for an optimal solution by moving the particle in the solution space, relative to its current position, at a certain velocity and guided by a fitness function: a particle's movement is controlled by changing its velocity (accelerating), guided by its current best position (pbest) and the best position found in its neighborhood (gbest)up to that point. The gbest solution is iteratively updated as better positions are found. The combined (collective) exploration of all the particles moves the swarm toward the best solution. In their quest for the global optimum, particles in the swarm realign to cluster around the suboptimal. Once a particle is close to the global optimum, other particles are attracted to it, with a high probability of finding the best solution. In each iteration *k*, particle *i* updates its position and velocity, according to the following equations:

$$v_i^{k+1} = x_i^{k+1} - x_i^k \tag{6-9}$$

$$x_i^{k+1} = x_i^k + v_i^k + c_1 r_1 (pbest_i^k - x_i^k) + c_2 r_2 (gbest_i^k - x_i^k) \tag{6-10}$$

$$v_i^{k+1} = v_i^k + c_1 r_1 (pbest_i^k - x_i^k) + c_2 r_2 (gbest_i^k - x_i^k), \tag{6-11}$$

where

x_i^k = particle *i* position for the *k*th iteration

v_i^k = particle *i* velocity for the *k*th iteration

c_1, c_2 = weighting coefficients

r_1, r_2 = random numbers between 0 and 1

The PSO algorithm is composed of the following steps for iteration *k*:

1. Initialize the swarm by allocating a random position x_i^0 to each particle *i* of the swarm bounded by the problem space.
2. Evaluate the fitness of each particle *i*, relative to its current position x_i^k.
3. Compare the particle *i* fitness with its $pbest_i^{k-1}$; if the current fitness is greater than the pbest, set the pbest value ($pbest_i^k$) to the current fitness value.
4. Select the particle *j* with the best fitness ($pbest_j^k$); mark this fitness $gbest_i^k$.
5. Evaluate the new position x_i^{k+1} of particle (*i*), using Equations 6–10.
6. Evaluate the new velocity v_i^{k+1} of particle (*i*), using Equations 6–11.

The process repeats until the stopping criteria are met, or the best solution is found. Unlike evolutionary algorithms, such as GA, particles improve the PSO algorithm's fitness, using the current global optimum, without evolutionary operators.

Because of its distributed nature and ability to operate under noisy conditions, PSO can prove to be a useful technique for workload balancing, with respect to power consumption, heat generation, and QoS requirements in cloud computing. The workload has to be distributed in such a manner that power consumption is minimized, thermal hot spots are eliminated, and performance targets are fulfilled. Dynamic placement of the workload in a system (or cluster of compute machines) triggers dynamic variations in the availability of compute, memory, network, input/output (I/O), and storage resources. Optimal system operation results in complex workload distribution choices, owing to the many degrees of freedom for allocating the load in a dynamically varying resource pool. The PSO solution continuously searches for the dynamically shifting optimum to identify the placement target of the new or upcoming load.

Yassa et al. (2013) proposed *DVFS multiobjective discrete particle swarm optimization* (DVFS-MODPSO) for workload scheduling in a distributed environment. DVFS-MODPSO implements the multiobjective optimization of several conflicting goals—minimizing execution time, execution cost, and energy consumption—and produces a set of nondominated solutions to offer flexibility in choosing a schedule that meets the QoS targets. DVFS-MODPSO defines a triplet $<T_i, P_j, V_k>$ that characterizes the position of a particle and that represents a reasonable solution to the workload scheduling problem. Each triplet allocates the task T_i to a processor P_j with a voltage scaling V_k. The results demonstrate that DVFS-MODPSO generates a set of Pareto optimal solutions for execution time, execution cost, and power consumption.

Solving a global optimization problem using a traditional approach involves precise function description and gradient evaluation, which may be expensive, time-consuming, hard to achieve, or impossible. Compounding the problem, many complex optimization problems exhibit a noisy behavior that renders methods such as implicit filtering and evolutionary gradient search almost ineffective. In contrast, PSO algorithms operate in a stable and efficient manner, even in the presence of noise. In many cases, noise can be beneficial, because it helps avoid local minimum solutions and converge faster to the globally optimal solution. Owing to their simplicity, PSO algorithms have also been proposed as an alternative to gradient-based techniques for detecting Pareto optimal solutions to multiobjective optimization problems.

Artificial Bee Colony Algorithm

The *artificial bee colony* (ABC) algorithm is a swarm-based metaheuristic inspired by the foraging behavior of the honeybee that was proposed by Karaboga, Dervis, and Basturk (2007). The model consists of three groups of honey bees that facilitate an optimal search for food sources. *Employed bees* attach themselves to a specific food source and share the information regarding its profitability through waggle dancing to recruit new bees. An *onlooker bee* is an unemployed bee that evaluates the quality of the food source by observing the waggle dances on the floor and deploys itself toward the most profitable food source. A *scout bee* searches for new food sources randomly and presents information associated with their quality through a waggle dance. The employed bee whose food source has been exhausted transforms itself into a scout bee and searches for new food sources. The principal components of the ABC algorithm are as follows:

- *Food sources*: A food source represents the candidate solution to an optimization problem. To select an optimal food source, an employed bee evaluates the overall quality of a food source, as measured by its proximity to the hive, the quantity and quality of the food (nectar), and the level of difficulty in extracting the food.
- *Employed bees*: Employed bees are employed at a specific food source, which they exploit to gather nectar. The bees collect information related to the distance, direction, and quality of the food source and share it with other bees, waiting on the dance floor. An employed bee attempts to improve its solution (food source) by reevaluating the coordinates in the neighborhood of its memorized coordinates, using multiple trials.
- *Unemployed bees*: Scout bees and onlooker bees are both in this category. They evaluate the profitability of potential food sources, either through random scouting or through information shared by employed bees. This evaluation helps convert an unemployed bee to an employed bee by facilitating selection of the most profitable food source.
- *Measure of quality (fitness)*: The quality of the food source—characterized by its proximity to the hive, the quantity and quality of its nectar, and the relative difficulty of extracting the nectar—can be summarized, using a single quantity: *fitness*.
- *Knowledge exchange*: Knowledge exchange is the critical element of the ABC algorithm. Knowledge is shared within the staging area, called the dance area; here, bees exchange information related to the fitness and coordinates (angle, distance) of the food source through the waggle dance.

The ABC algorithm can be used as a technique for load balancing in a datacenter. Load balancing attempts to optimize resource utilizationresponse time, throughput, and thermal hot spots. Load balancing can be implemented by reallocating existing tasks or allocating new tasks to an existing compute node. These compute nodes act as potential candidates for hosting the workload, which, when loaded effectively, can improve the efficiency of the datacenter. Each compute node advertises its prevailing characteristics (or fingerprint) related to utilization, operational phase, time spent in that phase, cache behavior, temperature, and power consumption. The fitness function defines a compute node's ability to host new work at a future time. For example, a candidate node that can compensate for the forecasted thermal variance in its neighborhood will have a higher fitness, compared with other nodes with similar characteristics but existing in a fully balanced cluster. Each server consists of management microcontrollers that act as idle, employed, onlooker, or scout bees. Scout bees are appointed in a random manner, whereas employed and onlooker bees follow the swarm behavior that is influenced by the fitness outcome. While an employed bee (management node) records the benefits of hosting the load on the existing node, an onlooker bee (waiting in a work queue), in its effort to become employed, analyzes the collective information delivered through scout and employed bees. Once the compute target is selected, the onlooker bee attempts to host the queued work on that target. The technique of load balancing using ABC deploys the following agents:

- *Scout bee*: Acts as a random agent that constitutes approximately 2 percent of the total compute nodes in a datacenter. These agents execute the scout function, using the management agent corresponding to the compute node tagged as scout bee. Scout agents collect neighborhood-specific information related to hot spots, average power consumption, and availability of compute resources.
- *Employed bee*: Acts as an agent that assists in loading and collecting the operational statistics of the load that is executing on the compute node tagged as employed bee. These statistics include usage, memory bandwidth, noisy behavior of the cache, and I/O contention.
- *Onlooker bee*: Acts as an agent of the potential workload waiting in the queue. Each agent identifies the best target to host this workload.

Two principal factors that attract bees to a specific node or neighborhood, in this example, are the availability of thermal variance and compute resources. As the thermal variance or compute resource diminishes, that node becomes less attractive and is eventually abandoned. While a node remains attractive, an employed bee repeatedly visits that location and encourages onlooker bees to host work in its neighborhood. Scout bees identify additional targets or neighborhoods with high fitness that can be exploited by unemployed bees.

The main steps of the ABC algorithm are generalized as follows:

1. Random food sources are allocated to each employed bee. Repeat:
    a. Each employed bee visits the food source, according to the information stored in the bee's memory. The bee evaluates the quantity and quality of the food (nectar) and performs the waggle dance in the hive.
    b. Each onlooker bee observes the waggle dance of the employed bees, and some of them select the food source, based on the information communicated through the dance.
    c. Once the food source is abandoned, new sources are identified by the scout bees.
    d. The new food source is identified by the scout bees and attracts the swarm, depending on the quality of the nectar.
    e. Requirements are met.
2. Repeat.

Bacterial Foraging Optimization Algorithm

The *bacterial foraging optimization* (BF0) algorithm (Passino 2002) models the microbiological phenomenon of organized behavior in a bacterial colony. The BFO algorithm for modeling the social foraging behavior of *Escherichia coli* (*E. coli*) can be used to solve real-world numeric optimization problems. BFO is primarily composed of three processes: chemotaxis, reproduction, and elimination–dispersal.

Chemotaxis is defined as cell movement in response to a chemical stimulus. This method is used by many single- and multicellular organisms to discover their food. Bacterial chemotaxis represents the signal transduction system, which stimulates the behavior of bacterial movement. *Reproduction* characterizes natural selection, which favors the best-adapted bacteria with a higher likelihood of survival than the less-adapted bacteria. Natural selection allows the selected population in each generation to transfer the genetic material to successive generations. *Elimination–dispersal* promotes the low probability of the elimination and dispersal of randomly selected parts of the bacterial population. This fosters diversity in the bacterial population and prevents the global optimal solution from being trapped in a local minimum.

E. coli alternates between two modes of movement, called *swim* and *tumble*, throughout its entire life. *Swimming* action allows the bacterium to move in the current direction of increasing nutrient gradient; *tumble* action allows change in orientation when the nutrient gradient is no longer attractive. The alternating mode of bacterial movement enables a bacterium to locate the position of the optimal nutrient source. After a certain number of complete swims, the bacterial population undergoes reproduction and elimination, according to the fitness criteria. Each bacterium position has an associated cost and represents a possible solution. BFO simulation keeps track of the cost of current and previous positions to estimate the quality of gradient improvement or worsening. In each generation the health of the bacterium factors into its likelihood of being retained for reproduction (making replicas) or elimination.

If θ^i represents the position of the *i*th bacteria, the successive movement of the bacterium is

$$\theta^i(j+1,k,l)=\theta^i(j,k,l)+v^i(j)\cdot\phi^i(j), \tag{6-12}$$

where

$\theta^i(j,k,l)$ = position of the *i*th bacterium in the *j*th chemotactic, *k* th reproductive, and *l*th elimination–dispersal step

$v^i(j)$ = step size in a random direction during the *tumble*

$\phi^i(j)$ = random direction of movement after the tumble

For the given position of the *i*th bacterium, $\theta^i(j,k,l)$, $J^i(j,k,l)$ represents the fitness of the bacterium at that location. If the fitness of *i*th bacterium at location $\theta^i(j+1,k,l)$ is better than that at $\theta^i(j,k,l)$, such that $J^i(j+1,k,l)$ is better than $J^i(j,k,l)$, then $v^i(j+1) = v^i(j)$, and $\phi^i(j+1) = \phi^i(j)$. If the reverse is true, then $v^i(j+1)$ takes a different step, in a random direction.

Munoz, Lopez, and Caicedo (2007) proposed a BFO algorithm for searching the best actuators in each sample time to obtain a uniform temperature over the temperature grid platform. The idea is to compensate for the cold spots by allocating or deallocating additional resources. Similar techniques can be applied to load balancing in a cluster of compute servers, as in a datacenter. Server load balancing techniques employ bacterial searches to locate the regions of nonuniform thermal behavior (high temperature variance). The fitness of the newly identified location can be evaluated by modeling the thermal variance in the temperature grid resulting from adding or subtracting quantities of unit load.

Artificial Immune System

The *artificial immune system* (AIS) is a bioinspired optimization algorithm (Dasgupta 1999) based on the principles of the vertebrate immune system. The algorithm emulates several characteristics of the human immune system: that it is highly distributed, that it is parallel, and that it uses adaptive learning and memory to solve problems related to pattern recognition and classification. The AIS algorithm learns to categorize relevant patterns through a pattern detector that associates previously -seen patterns with existing ones. The algorithm formulates a different response mechanism to deal with the effects of each pattern.

The adaptive immune system in the human body uses many agents that perform diverse functions at different locations, primarily employing negative selection and clonal selection mechanisms. *Negative selection mechanisms* exploit the immune system's ability to detect unknown antigens, while not reacting to self. *Clonal selection mechanisms* promote the proliferation of cells that possess the ability to recognize an antigen over those that do not. Therefore, self-reacting cells are eliminated, and mature cells are allowed to proliferate. The learning mechanism involves bolstering by the cloning process of those lymphocytes within a given population that contribute to the identification of an antigen. New cells are copies (*clones*) of their parents, but cloning is subject to a high rate of mutation (*somatic hypermutation*). This mutation process mimics the mechanism that reallocates the resources needed for recognition of new antigens versus previously identified antigens. The reinforced learning mechanism rebalances the population of diverse lymphocytes to promote optimal detection and mediation of pathogens.

The properties of the immune system have the following attributes (Castro, Nunes, and Von Zuben 1999):

- *Exclusivity*: The immune system is exclusive to each individual, with its own vulnerabilities and capabilities.
- *Recognition of foreigners*: The toxic elements or molecules that are foreign to the individual's body are identified, categorized, and labeled for future detection.
- *Anomaly detection*: The immune system learns to classify the unidentified foreign element as a pathogen and attempts a remedial action.
- *Distributed detection*: The cells are distributed throughout the body and are not subject to centralized control.
- *Imperfect detection* (*noise tolerance*): Pathogens are first classified as unidentified foreign elements, and their absolute recognition is not essential.
- *Reinforcement learning and memory*: The immune system continuously learns the structure of the pathogen to formulate an increasingly effective response.

Similar to that of the GA, AIS architecture comprises the following four steps (Aickelin, Dasgupta, and Gu 2014):

1. *Encoding*: Encoding is binary, numeric, or nominal representation of antigens or antibodies. An *antigen* represents the solution to a problem domain that needs to be tested for an intrusion. Antibodies represent previously identified patterns that can be used later.
2. *Similarity measure*: A similarity measure quantifies the *affinity* between an antigen and its candidate antibodies. The matching algorithm measures the extent of *agreement*, *disagreement*, or *correlation* between a candidate antibody and its target antigen. Candidates with strong agreement or disagreement may be selected for further processing (cloning or mutation).

3. *Selection*: The selection process follows an iterative procedure, in which the concentration of antibodies is regulated by cloning or removal at each step, depending on the antibody–antigen affinity measure. Upon adding a new antibody, the iterative process changes the concentration of that antibody, continuing until the AIS achieves stability. AIS iteration can be represented by the following equation (Farmer, Packard, and Perelson 1986):

$$\dot{x}_i = k_1\left(\sum_{j=1}^{n} m_{ji}x_iy_j - k_2x_i\right), \tag{6-13}$$

where

n = number of antigens

x_i = concentration of antibody i

y_j = concentration of antigen j

m_{ji} = affinity function representing the correlation between antibody i and antigen j

k_1 = rate of antibody production

k_2 = death rate

Equation 6-13 represents the iterative change in the antibody concentration, contingent on the net outcome of cloning due to antigen recognition and death in the absence of correlation.

4. *Mutation*: Antigen–antibody interaction, coupled with somatichypermutation, forms the basis of an AIS. Mutation introduces diversity in the population and facilitates effective response to antigens.

AIS uses an adaptive population of antibodies to facilitate intelligent behavior by synthesizing diverse subset solutions for a given problem domain. AIS has been applied in areas related to network security and anomaly detection.

Distributed Management in Datacenters

Datacenters are complex environments that deal with key challenges related to power delivery, energy consumption, heat management, security, storage performance, service assurance, and dynamic resource allocation. These challenges relate to providing effective coordination to improve the stability and efficiency of datacenters. The fluctuating demands and diverse workload characteristics of a large datacenter make complex the tasks of upholding workload performance, cooling efficiency, and energy targets (discussed in the following sections). In such large clusters of systems, multiple objectives compete to accomplish service-level goals by avoiding actuator overlapping and exhausting a complex combination of constraints, timing granularity, type of approach, and sequence of controls. However, the combinatorial solution space can be extremely large and may not converge to a global optimal in a bounded time. Therefore, a centralized datacenter management system may not scale well in constrained time and hence may not deliver an optimal management solution.

SI has emerged as a promising field that can be exploited to construct a distributed management methodology leading to scalable solutions without centralized control. The following sections present a control system that identifies suitable targets for workload placement, with these fundamental control elements:

- *Controlled process*: The controlled process implements the feedback control loop, which constrains the temperature and power of compute clusters, such as server racks, for a given policy. An optimal process operates within policy constraints and provisions sufficient energy to operate a workload at highest performance efficiency and lowest cooling.
- *Fitness function*: The fitness function estimates the most favorable placement of the workload, based on the existing knowledge base's expected demand and availability of resources.
- *Knowledge base*: The knowledge base acts as a finite database made up of survey data conducted by the sensor agents. This knowledge assists in identifying the most probable placement of the workload. As the dynamics of the system changes, newer data replace the old data, according to a custom data retention policy. The knowledge database increases the retention of data likely to boost the fitness of the solution and deprecates the data less likely to improve the existing solution.
- *Control parameters*: Control parameters define the optimal decision boundaries that result in placement of the workload on the selected compute node.
- *Swarm agents*: Swarm agents participate in the system optimization process by executing specific roles in a decentralized and self-organized system. These agents coordinate with each other and with the environment, ultimately leading to the emergence of intelligent global behavior.

Workload Characterization

Because workloads undergo phases of execution, phase boundaries are fundamental attributes for predicting workload behavior for scheduling or migration between clusters of servers. Additionally, phase identification enables reuse of past configurations of recurring phases to improve performance. These configurations enforce a policy for scheduling new workloads, migrating existing workloads, and eliminating thermal load imbalances between compute nodes or clusters of compute nodes.

Thermal Optimization

Given the highly dynamic environment in a datacenter, hot spots are created as a result of temporal events (such as increased workload on a set of servers) or spatial events (such as inefficiency of the computer room air-conditioning [CRAC]) units in delivering the requisite cooling to a particular region in the datacenter). Figure 6-2 depicts a thermal snapshot of a datacenter with hot and cold spots. Hot spots may trigger overcooling, degrading the *power usage effectiveness* (PUE) of the datacenter operations. Traditional cooling control solutions operate using reactive schemes, which depend on the instantaneous temperatures of racks or blades. These schemes have a fundamental disadvantage, inasmuch as the corrective action is completed long after the component's thermal or performance threshold has been crossed. In a datacenter with a large number of nodes, it is almost impossible to perform optimal workload balance in real time using a reactive approach without causing hysteresis. In the presence of dynamic variations in a cluster of configurable hardware and software, the ability to initiate a timely response to reduce temperature variance (hot spots) between clusters is essential.

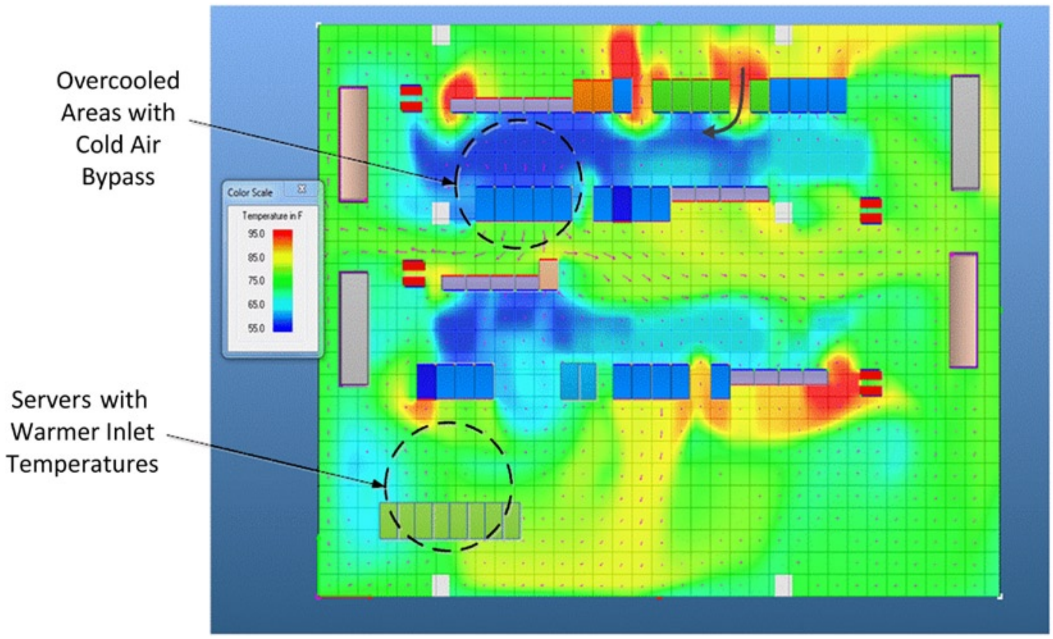
Overcooled Areas with Cold Air Bypass
Servers with Warmer Inlet Temperatures
Color Scale
Temperature in F
95.0
85.0
75.0
65.0
55.0

Figure 6-2. *Thermal snapshot of a datacenter*

Load Balancing

Load balancing is a method for distributing workload among a cluster of compute nodes in a manner that allows fulfillment of a given policy. Load balancing optimizes resource usage, maximizes throughput, and minimizes response time. Load balancing is realized through active load migration between compute nodes, which also governs the hot-spot mitigation scheme, using workload tradeoffs and power distribution. Hot spots can be attributed to uneven load distributions leading to imbalanced compute utilization. Hot spots result in inefficient cooling and higher datacenter operating costs. In a swarm-based optimization scheme thermal variance can be equated with foraging for a source of nutrients. Regions of high thermal variance act as a source (or target) of load migration, attracting increased surveillance from the swarm agents.

Algorithm Model

The *algorithm model* consists of a server manager, a scout agent, worker agents, free agents, and a load controller. The *server manager* administers each server node and presents programming interfaces for interpreting the sensor data and synthesizing the useful metrics. The server manager exerts control at multiple levels of timing granularity, which can eventually result in heterogeneous sampling requirements specific to each one of those elements. For instance, the server manager records the performance data and processes them to synthesize the workload phase distribution by exercising built-in sensors. In a hierarchical scheme the server manager manages a cluster of compute resources and identifies the compute resource capable of hosting a candidate workload. Externally, each cluster represents a set of compute units that is managed locally, without exposing its local hierarchy.

Scout agents are randomly generated proxy instruments that evaluate their surroundings and peer clusters to identify possible sources of hot and cold spots. Owing to the dynamic nature of the systemutilization, newer sources are identified, and past sources are slowly forgotten. Scout agents employ fitness criteria to determine the suitability of the region to host the work assignment.

Worker agents are floating entities that use a feedback function to quantify the significance of the compute region and that attach themselves to a node that they select for foraging. At each scheduling instant a feedback function measures the historical impact by either boosting or shedding the node's credibility, based on the most recent scheduling decisions and preceding outcomes. If the compute region continues to host workloads successfully, it boosts its credibility; if, however, the compute region loses its bid to host or hosts infrequently, then the credibility declines. Worker agents are finite in number and can only be reused if one is released, owing to either its low impact score or its migration to newer regions. The feedback function is

$$\beta_{n+1}^{k} = (1-\rho)\beta_{n}^{k} + \tau_{n}^{k}, \tag{6-14}$$

where

β_n^k = impact score of cluster (or node) *k* at instance *n*

τ_n^k = incremental credit boost of cluster (or node) *k* at instance *n*

ρ = impact decay coefficient

As described in function 6-14, whenever a compute cluster or node hosts a workload successfully, it improves its impact score by receiving incremental credits. At the same time, the feedback function sheds a percentage of its score at a rate defined by the impact decay coefficient ρ. The decay coefficient lets the rate of adaptation be adjusted. At any one time, only a finite number of worker agents is allowed to operate. A worker agent transitions to a *free agent* when its fitness score falls below a certain threshold. Once this occurs, the worker agent can attach itself to a new cluster, based on feedback from scout agents and worker agents.

Whereas scout agents identify new and promising regions for exploration, worker agents help in characterizing desirable neighborhoods close to their current operating region. At all times, these agents evaluate the cluster's fitness, a measure of its ability to host a new workload. This fitness score allows the respective agents to participate in a bidding process to host the workload in the region they represent. The fitness evaluation assesses the following characteristics:

- Severity of the thermal imbalances in the attached region, as compared with other regions
- Availability of the resources needed to execute a workload successfully
- Degree of contention with respect to available resources shared among compute nodes (e.g., shared cache)

Figure 6-3 depicts the architectural interfaces of a compute node that facilitate optimal workload distribution. Worker and scout agents gather node-specific performance and environmental data, using instrumentation APIs provided by the server manager. The agents explore the viability of using the node to host a workload by measuring the node's impact score (see Equation 6-14), evaluating its resource requirements, forecasting shared resource contention, and predicting the temperature behavior that may lead to thermal imbalances. Once the node is ascertained to be a potential host, its impact score is updated for future analysis. Cold regions with sufficient availability of resources identify themselves as the preferred localities for exploration. Worker agents analyze the compatibility of each region with the workload; incompatibilities may arise, owing to lack of exclusive resources or historical evidence of noisy behavior, with respect to the sharing of resources with other workloads. The worker agent ranks the host it is attached to and makes a decision as to whether to participate in the bidding process to host the workload on that node. If it participates and wins the bid, the agent updates the feedback function; if it does not win the bid, the agent sheds a percentage of its impact score *b*. Once the impact score drops below a certain threshold, the worker agent transitions to a free agent. A free agent transitions its role back to worker agent by attaching itself to a new cluster (or node), based on an evaluation score received from the scout agents and worker agents.

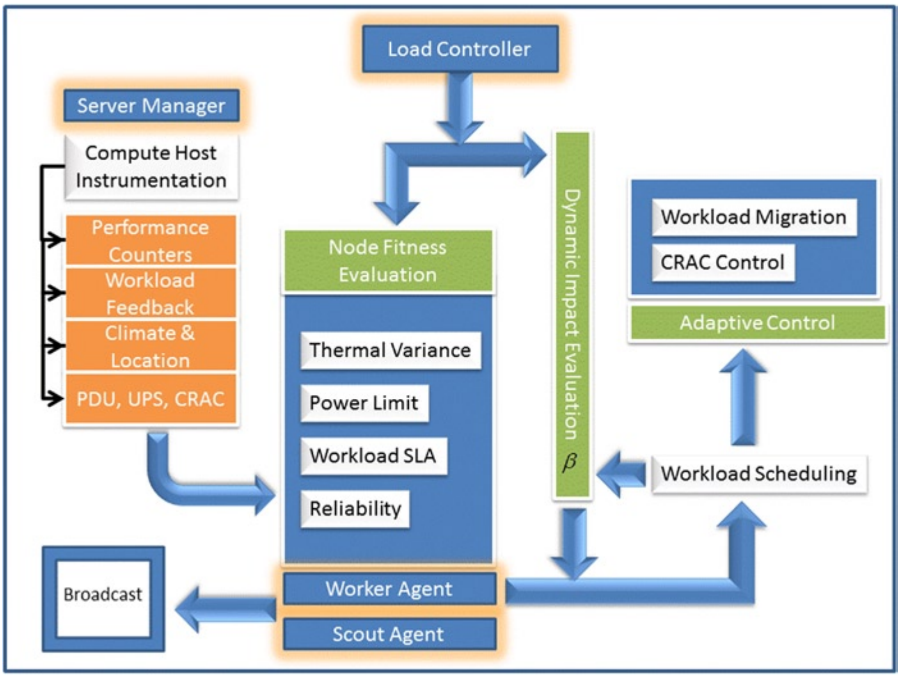

Figure 6-3. *Compute node: architectural interfaces for interpreting sensor data for evaluating feedback function and fitness function*

References

Aickelin, Uwe, Dipankar Dasgupta, and Feng Gu. "Artificial Immune Systems." In *Search Methodologies*, 187–211. New York: Springer, 2014.

Ashby, W. Ross. "Design for a Brain." New York: Wiley, 1952. https://archive.org/details/designforbrainor00ashb.

Barbagallo, Donato, Elisabetta Di Nitto, Daniel J. Dubois, and Raffaela Mirandola. "A Bio-Inspired Algorithm for Energy Optimization in a Self-Organizing Data Center." In *Self-Organizing Architectures: First International Workshop, SOAR 2009, Cambridge, UK, September 14, 2009*, edited by Danny Weyns, Sam Malek, Rogério de Lemos, and Jesper Andersson, 127–151. Berlin: Springer, 2010.

Beni, Gerardo, and Jing Wang. "Swarm Intelligence in Cellular Robotic Systems." In *Robots and Biological Systems: Towards a New Bionics?*, edited by Paolo Dario, Giulio Sandini, and Patrick Aebischer, 703–712. Berlin: Springer , 1993.

Carreras, I., D. Miorandi, G. S. Canright, and K. Engo-Monsen. "Understanding the Spread of Epidemics in Highly Partitioned Mobile Networks." In *Proceedings of the 1st IEEE Conference on Bio-Inspired Models of Network, Information and Computing Systems*, 1–8. Piscataway, NJ: Institute of Electrical and Electronics Engineers, 2006. Dasgupta, Dipankar. *Artificial Immune Systems and Their Applications.* Berlin: Springer, 1999.

de Castro, Leandro Nunes, and Fernando José Von Zuben. "Artificial Immune Systems: Part I–Basic Theory and Applications." Technical report, Universidade Estadual de Campinas, 1999.

Devescovi, Davide, Elisabetta Di Nitto, Daniel Dubois, and Raffaela Mirandola. "Self-Organization Algorithms for Autonomic Systems in the SelfLet Approach." In *Proceedings of the 1st International Conference on Autonomic Computing and Communication Systems*. Brussels: Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, 2007.

Di Caro, Gianni, and Marco Dorigo. "AntNet: Distributed Stigmergetic Control for Communications Networks." *Journal of Artificial Intelligence Research* 9 (1998): 317–365. www.cs.cmu.edu/afs/cs/project/jair/pub/volume9/dicaro98a.pdf.

Di Caro, Gianni, Frederick Ducatelle, and Luca Maria Gambardella. "AntHocNet: An Adaptive Nature-Inspired Algorithm for Routing in Mobile Ad hoc Networks." *European Transactions on Telecommunications* 16, no. 5 (2005): 443–455.

Dorigo, Marco, Gianni Di Caro, and Luca Gambardella. "Ant Algorithms for Discrete Optimization." *Artificial Life* 5, no. 2 (1999): 137–172.

Engelbrecht, Andries P. *Fundamentals of Computational Swarm Intelligence*. Chichester, UK: Wiley, 2006.

Farmer, J. Doyne, Norman H. Packard, and Alan S. Perelson. "The Immune System, Adaptation, and Machine Learning." *Physica D: Nonlinear Phenomena* 22, no. 1 (1986): 187–204.

Harvey, Inman, Ezequiel Di Paolo, Rachel Wood, Matt Quinn, Elio Tuci, and Elio Tuci. "Evolutionary Robotics: A New Scientific Tool for Studying Cognition." *Artificial Life* 11, no. 1–2 (2005): 79-98.

Iyer, Ravi. "CQoS: A Framework for Enabling QoS in Shared Caches of CMP Platforms." In *Proceedings of the 18th Annual International Conference on Supercomputing*, 257–266. New York: ACM, 2004.

Karaboga, Dervis, and Bahriye Basturk. "Artificial Bee Colony (ABC) Optimization Algorithm for Solving Constrained Optimization Problems." In *Proceedings of the 12th international Fuzzy Systems Association World Congress on Foundations of Fuzzy Logic and Soft Computing, IFSA 2007, Cancun, Mexico, June 18–21, 2007,* edited by Patricia Melin, Oscar Castillo, Luis T. Aguilar, Janusz, Kacprzyk, and Witold Pedrycz, 789–798. Berlin: Springer, 2007.

Kennedy, J., and R. Eberhart. "Particle Swarm Optimization." In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, 1942–1948. Piscataway, NJ: Institute of Electrical and Electronics Engineers, 1995.

Khanna, Rahul, Huaping Liu, and Hsiao-Hwa Chen. "Reduced Complexity Intrusion Detection in Sensor Networks Using Genetic Algorithm." In *Proceedings of the 2009 IEEE International Conference on Communications*, 1–5. Piscataway, NJ: Institute of Electrical and Electronics Engineers, 2009.

Jacob, Bart, Richard Lanyon-Hogg, Devaprasad K. Nadgir, and Amr F. Yassin. "A Practical Guide to the IBM Autonomic Computing Toolkit." Armonk, NY: IBM, 2004. www.redbooks.ibm.com/redbooks/pdfs/sg246635.pdf.

Li, Zhen, and Manish Parashar. "Enabling Autonomic Grid Applications: Dynamic Composition, Coordination and Interaction." In *Unconventional Programming Paradigms: Proceedings of the International Workshop UPP 2004, Le Mont Saint Michel, France, September 2004; Revised and Selected Papers*, edited by Jean-Pierre Banâtre, Pascal Fradet, Jean-Louis Giavitto, and Olivier Michel, 270–285. Berlin: Springer, 2005.

Munoz, Mario A., Jesus A. Lopez, and Eduardo Caicedo. "Bacteria Swarm Foraging Optimization for Dynamical Resource Allocation in a Multizone Temperature Experimentation Platform." In *Analysis and Design of Intelligent Systems Using Soft Computing Techniques*, 427–435. Berlin: Springer, 2007.

Papadopouli, Maria, and Henning Schulzrinne. "Seven Degrees of Separation in Mobile Ad Hoc Networks." In *Proceedings of the 2000 IEEE Global Telecommunications Conference*, 1707–1711. Piscataway, NJ: Institute of Electrical and Electronics Engineers, 2000.

Passino, Kevin M. “Biomimicry of Bacterial Foraging for Distributed Optimization and Control.” *IEEE Control Systems* 22, no. 3 (2002): 52–67.

Schoonderwoerd, Ruud, Owen E. Holland, Janet L. Bruten, and Leon JM Rothkrantz. “Ant-Based Load Balancing in Telecommunications Networks.”*Adaptive Behavior* 5, no. 2 (1997): 169–207.

Wang, Xiaolei, Xiao Zhi Gao, and Seppo J. Ovaska. “A Hybrid Optimization Algorithm Based on Ant Colony and Immune Principles.” *International Journal of Computer Science and Applications* 4, no. 3 (2007): 30–44.

Yassa, Sonia, Rachid Chelouah, Hubert Kadima, and Bertrand Granado. “Multi-Objective Approach for Energy-Aware Workflow Scheduling in Cloud Computing Environments.” *Scientific World Journal* 2013 (2013): 350934. www.hindawi.com/journals/tswj/2013/350934/.