J Grid Computing (2011) 9:279–301
DOI 10.1007/s10723-011-9180-5

# Swarm Intelligence Approaches for Grid Load Balancing

**Simone A. Ludwig · Azin Moallem**

Received: 3 August 2010 / Accepted: 2 February 2011 / Published online: 22 February 2011

**Abstract** With the rapid growth of data and computational needs, distributed systems and computational Grids are gaining more and more attention. The huge amount of computations a Grid can fulfill in a specific amount of time cannot be performed by the best supercomputers. However, Grid performance can still be improved by making sure all the resources available in the Grid are utilized optimally using a good load balancing algorithm. This research proposes two new distributed swarm intelligence inspired load balancing algorithms. One algorithm is based on ant colony optimization and the other algorithm is based on particle swarm optimization. A simulation of the proposed approaches using a Grid simulation toolkit (GridSim) is conducted. The performance of the algorithms are evaluated using performance criteria such as makespan and load balancing level. A comparison of our proposed approaches with a classical approach called State Broadcast Algorithm and two random approaches is provided. Experimental results show the proposed algorithms perform very well in a Grid environment. Especially the application of particle swarm optimization, can yield better performance results in many scenarios than the ant colony approach.

S. A. Ludwig (✉) · A. Moallem
Department of Computer Science, North Dakota State University, Fargo, ND, USA
e-mail: simone.ludwig@ndsu.edu

**Keywords** Ant colony optimization · Particle swarm optimization

## 1 Introduction

The computational speed of individual computers has increased by about one million times in the past fifty years. However, they are still not fast enough for more and ever more scientific problems. For example, in a few physics applications, data is produced in large quantities. The analysis of this data would need much more computational power than presently available when run on supercomputers. Therefore, in the mid 1990s Ian Foster and Carl Kesselman proposed a distributed computing infrastructure for advanced science and engineering, which they called the Grid. The vision behind the Grid is to supply computing and data resources over the Internet seamlessly, transparently and dynamically when needed, such as the power Grid supplies electricity to end users.

The resource management system is the central component of a Grid system. Its basic responsibilities are to accept requests from users, match user requests to available resources for which the user has permission to use and schedule the matched resources [12]. To be able to fully benefit from such Grid systems, resource management and scheduling are key Grid services, where issues of task allocation and load balancing represent a common challenge for most Grids [27]. In a computational Grid, at a given time, the task is to allocate the user defined jobs efficiently both by meeting the deadlines and making use of all the available resources [10].

Grid systems are classified into two categories: *compute* and *data Grids*. In *compute Grids* the main resource that is being managed by the resource management system is compute cycles (i.e. processors), while in *data Grids* the focus is to manage data distributed over geographical locations. The architecture and the services provided by the resource management system are affected by the type of Grid system it is deployed in. Resources which are to be managed could be hardware (computation cycle, network bandwidth and data stores) or software resources (applications) [12].

In traditional computing systems, resource management is a well-studied problem. Resource managers such as batch schedulers, workflow engines, and operating systems exist for many computing environments. These resource management systems are designed to work under the assumption that they have complete control of a resource and thus can implement the mechanisms and policies needed for the effective use of that resource. Unfortunately, this assumption does not apply to the Grid. When dealing with the Grid we must develop methods for managing Grid resources across separately administered domains, with the resource heterogeneity, loss of absolute control, and inevitable differences in policy that is the result of heterogeneity. The underlying Grid resource set is typically heterogeneous [9].

The term "load balancing" refers to the technique that tries to distribute work load between several computers, network links, CPUs, hard drives, or other resources, in order to get optimal resource utilization, throughput, or response. The load balancing mechanism aims to equally spread the load on each computing node, maximizing their utilization and minimizing the total task execution time. In order to achieve these goals, the load balancing mechanism should be "fair" in distributing the load across the computing nodes; by being "fair" we mean that the difference between the "heaviest-loaded" node and the "lightest-loaded" node should be minimized [20].

Load balancing has always been an issue since the emergence of distributed systems. In a distributed system there might be scenarios in which a task waits for a service at the queue of one resource, while at the same time another resource which is capable of serving the task is idle. The purpose of a load balancing algorithm is to prevent these scenarios as much as possible [16].

For parallel applications, load balancing attempts to distribute the computational load across multiple processors or machines as evenly as possible with the objective to improve performance. Generally, a load balancing scheme consists of three phases: *information collection, decision making* and *data migration*. During the *information collection* phase, the load balancer gathers the information of the distribution of workload and the state of computing environment and detects whether there is a load imbalance. The *decision making* phase focuses on calculating an optimal data distribution, while the *data migration* phase transfers the excess amount of workload from one overloaded processor to another underloaded processor [15].

Load balancing algorithms can be classified into sub categories from various perspectives. They can be divided into *static*, *dynamic* or *adaptive* algorithms. In *static* algorithms, the decisions related to balancing the load are made at compile time. This means, these decisions are made when resource requirements are estimated [30]. On the other hand, a load balancer with *dynamic* load balancing allocates/re-allocates resources at runtime and uses the system-state information to make its decisions. *Adaptive* load balancing algorithms are a special class of *dynamic* algorithms. They adapt their activities by dynamically changing their parameters, or even their policies, to suit the changing system state [24]. Derbal [32] proposes a scalable distributed Entropy-based scheduling approach that utilizes a Markov chain model to capture the dynamics of the service capacity state of Grid services. Another example is proposed in [34] with the introduction of the Generational Scheduling with Task Replication algorithm. This algorithm adapts to changes in performance by rescheduling tasks.

Furthermore, methods used in load balancing can be divided into three classes, i.e., *centralized, distributed (decentralized)* and *hierarchical* [12]. In a *centralized* approach, all jobs are submitted to a single scheduler. This single scheduler is responsible for scheduling the jobs on the available resources. Since all the scheduling information is available at once, the scheduling decisions are optimal but this approach is not very scalable in a Grid system [12]. As the size of the Grid increases, keeping all the information about the state of all the resources is a bottleneck. Therefore, scalability is an issue in centralized approaches, in addition to the single point of failure problem.

In a *decentralized* model there is no central scheduler and scheduling is done by the resource requesters and owners independently. This approach is scalable, distributed in nature, and suits Grid systems well. But individual schedulers should cooperate with each other in scheduling decisions and the schedule generated may not be the optimal schedule. This category of load balancing is perfect for peer-to-peer architectures and dynamic environments. Based on whether or not schedulers cooperate with each other, decentralized approaches can be further classified as *cooperative* or *non-cooperative* [12]. A general and extensible scheduling architecture that addresses isssues such as resource utilitzation, response time, global and local allocation policies, and scalability are addressed in [33].

In a *hierarchical* model, the schedulers are organized in a hierarchy. High level resource entities are scheduled at higher levels and lower level smaller sub-entities are scheduled at lower levels of the scheduler hierarchy. This model is a combination of the above two models [12].

Each of these classes has its advantages and disadvantages according to a number of factors, e.g., the size of a system, dynamic behavior, etc. [31]. However, all centralized approaches have the following common disadvantages:

1. A central scheduler (load balancer) needs current knowledge about the entire state of the system at each point in time. This makes it scale poorly with the growth in the size of the system.
2. Failure of the scheduler results in failure of the whole system, while in a distributed approach only some of the work is lost.
3. Distributed schedulers are much more dynamic and flexible to changes than centralized approaches, because they do not need the state of the system at each step in order to perform their job.

There has been a great effort in recent years in developing distributed load balancing algorithms, while trying to minimize all the communication needs resulting from the distributed nature. In this research, we have focused on designing distributed load balancing algorithms with the inspiration taken from swarm intelligence.

Swarm intelligence approaches are increasingly being used to solve optimization problems. They have proven themselves to be good candidates in these areas. The notion of complex collective behavior emerging from the behavior of many relatively simple units, and the interactions between them, is fundamental to the field of swarm intelligence. The understanding of such systems offers new ideas in creating artificial systems which are controlled by such emergent collective behavior; in particular, the exploitation of this concept might lead to completely new approaches for the management of distributed systems, such as load balancing in Grids [23].

As swarm intelligence techniques have proved to be useful in optimization problems they are good candidates for load balancing, where the aim is to minimize the load difference between the *heaviest* and *lightest* node. The benefit of these techniques stems from their capability in searching large search spaces very efficiently, which arise in many combinatorial optimization problems [26]. Load balancing is known to be NP-complete when aiming to solve the problem using a single processor, therefore the use of heuristics is definitely necessary in order to cope in practice with this difficulty [10].

This research proposes, implements and compares two new approaches for distributed load balancing inspired by Ant-Colony and Particle Swarm Optimization [17]. There are several objectives a good load balancer should address such as fairness, robustness and distribution; a detailed description of each is provided in Section 3. These requirements are addressed with the design of our algorithms. In the Ant-Colony approach each job submitted to the Grid invokes an ant and the ant searches through the network to find the best node to deliver the job to. Ants leave information related to the nodes they have seen as pheromone in each node which helps other ants to find lighter resources more easily. In the particle swarm approach, each node in the network is considered to be a particle and tries to optimize its load locally by sending or receiving jobs to and from its neighbors. This process being done locally for each node, results in a move toward the global optima in the overall network.

The remainder of this paper is organized as follows: Section 2 is dedicated to related work. Load balancing algorithms that are decentralized are summarized. The requirements for the design of the distributed load balancing algorithms and the benefits are discussed in Section 3. It is followed by the proposed approaches which are described in detail. Section 4 focuses on the setup of the simulation and the experimental results. Performance criteria and environmental settings are introduced in this section, and a thorough comparison of the performance of the algorithms with other classical approaches is provided. Finally, Sections 5 and 6 are dedicated to conclusion and future work.

## 2 Related Work

This section will provide an account of related work only of decentralized approaches. Research in the area of distributed load balancing is diverse, and many researchers have used Ant colony for routing and load balancing.

### 2.1 Classical Approaches

There are several classical approaches in the area of load balancing which have been around since the emergence of networks. In *sender-initiated algorithms*, load distributing activity is initiated by an overloaded node (sender) trying to send a task to an underloaded node (receiver) [24]. In *receiver-initiated algorithms*, load distributing activity is initiated from an underloaded node (receiver), which tries to get a task from an overloaded node (sender) [24]. A *stable symmetrically initiated adaptive algorithm* uses the information gathered during polling (instead of discarding it, as the previous algorithms do) to classify the nodes in the system as sender/overloaded, receiver/underloaded, or OK (nodes having manageable load). The information about the state of the nodes is maintained at each node by a data structure composed of a senders list, a receivers list, and an OK list. These lists are maintained using an efficient scheme and list-manipulative actions, such as moving a node from one list to another, or determining to which list a node belongs. These actions impose a small and constant overhead, irrespective of the number of nodes in the system. Consequently, this algorithm scales well to large distributed systems [24].

The *Random* approach is a simple scheduling algorithm in which the jobs being sent to the Grid are assigned randomly to different resources. Although, obviously this approach does not make a very good load balancing algorithm but it has some benefits. It poses no decision making overhead on the system and it gives a good benchmark in order to compare and see how our proposed algorithms improve the performance of load balancing compared to a single random assignment.

The other approach we use to evaluate the performance of our proposed algorithms is the *State Broadcast Algorithm (SBA)*. This algorithm is common in networking, and is based on broadcast messages which are exchanged between resources. Whenever the state of a node changes, due to the arrival or departure of a task, the node broadcasts a status message that describes its new state. This information policy enables each node to hold its own updated copy of the System State Vector (SSV) and guarantees that all the copies are identical. When a job is sent to a resource at the time of scheduling, the resource searches through its own state vector to find the best resource available to deliver the job to at that particular time. SBA is a good benchmark to evaluate the performance of our algorithms as it resembles central approaches in which the status of the whole Grid is known at the time of scheduling, although being technically a distributed approach. SBA performs like central approaches, which by nature always outperform distributed ones [31], however, it has its disadvantages as mentioned earlier.

## 2.2 Ant Colony Optimization Approaches

Ant colony optimization has been widely used in both routing and load balancing [14]. Ant Colony Optimization (ACO) is considered a subset of social insect system approaches. The main idea underlying this approach is the indirect communication ability of ants depositing pheromone trails, which are then used by other ants.

One of the research work which is very similar to the ant colony algorithm we propose in this paper is the Messor system [18]. Montresor et al. have used an ant colony approach to develop a framework called Anthill, which provides an environment for designing and implementing peer-to-peer systems. They have developed Messor as a distributed load balancing application based on Anthill and have performed simulations to show how well Messor works. In the algorithm, the authors propose ants that can be in one of the two states: *Search-Max* or *Search-Min*. In the *Search-Max* state the ants try to find an overloaded node in the network and in the *Search-Min* state they search for underloaded nodes. Finally, the ants switch jobs between overloaded and underloaded nodes and hence achieve the balancing of the load. However, the authors have not addressed the problem of topology changes in the network and do not provide evidence to show how good their approach is in comparison to other distributed load balancing approaches.

In [4], a very similar approach to Messor is provided. In this work agent-based self-organization is proposed to perform complementary load balancing for batch jobs with no explicit execution deadlines. In particular, an ant-like self-organizing mechanism is introduced and is shown to be able to yield good results in achieving overall Grid load balancing through a collection of very simple local interactions. Ant-like agents move through the network to find the most overloaded and underloaded nodes, but the difference to previous research is they only search $2m + 1$ steps before making a decision and then they balance the load. Different performance optimization strategies are carried out, however, they do not compare their results with other distributed load balancing strategies.

Salehi and Deldari [19], have done similar research to [18] and [4] with some small modifications. They present an ecosystem of intelligent, autonomous and cooperative ants. The ants in this environment can reproduce offsprings when they realize that the system is unbalanced. They may also commit suicide when the equilibrium in the environment is reached. The ants wander $m$ steps instead of $2m + 1$ and they balance $k$ overloaded nodes and $k$ underloaded nodes instead of one at a time. A new concept called *Ant level load balancing* is presented for improving the performance of the mechanism. When the ants meet each other at the same node they exchange the information they carry with them, and continue on their way.

Sim et al. [14, 25], present a Multiple Ant Colony Optimization (MACO) for load balancing circuit-switched networks. In MACO, more than one colony of ants are used to search for optimal paths and each colony of ants deposits a different type of pheromone represented by a different colour. MACO optimizes the performance of a congested network by routing calls via several alternative paths to prevent possible congestion along an optimal path.

Another related and similar research to the ant colony approach we propose in this paper is done by Al-Dahoud et al. [2]. In their research, each node sends a coloured colony through the network; this approach helps in preventing ants of the same nest from following the same route, and hence, enforcing them to be distributed all over the nodes in the network. However, the authors' experimental results are confined to a small number of nodes and all the jobs have the same properties.

Martin Heusee et al. [11], have used multi-agent systems which have some similarity to ants to solve the problem of routing and load balancing in dynamic communication networks. They have proposed two kinds of routing agents depending on when the distance vector update occurs. The update can be performed while agents are finding their way to their destination (forward routing) or when they backtrack their way back to their source (backward routing).

Other similar research which benefits from the Ant colony's approach mostly focus on load balancing in routing problems [22] and [14]. In [14], the research provides a survey of four different routing algorithms: ABC, Ant-Net, ASGA. ABC is an Ant-Based Control system. A network with a typical distribution of calls between nodes is simulated, and nodes with an excess of traffic can become congested and can cause calls to be lost. Using the ant colony concept, the ants move randomly between nodes, selecting a path at each intermediate node based on the distribution of simulated pheromones at each node. As they move, they deposit simulated pheromones as a function of their distance from their source node, and the congestion encountered on their way [22]. In AntNet, they have applied ideas of the ant colony paradigm to solve the routing problem in datagram networks. Ants collect information about the congestion status of the followed paths and leave this information locally in the nodes. On the way back from the destination to the source, the local visiting table of each visited nodes are modified accordingly [5]. ASGA integrates ant colony systems with genetic algorithms. Each agent in the ASGA system encodes two parameters - the sensitivity to link and sensitivity to pheromone parameters. Each agent in the population has to solve the problem using an ant system and each agent has a fitness according to the solution found [29].

### 2.3 Summary

Existing decentralized approaches, which are mostly based on Ant colonies, are not accompanied with various performance measures to state how they perform in different scenarios and situations. Still, decentralized approaches in the Grid infrastructure are fewer in number than approaches designed for networks and peer-to-peer systems. In this research, we introduce two new load balancing algorithms, one based on Ant colony optimization and the other based on particle swarm optimization. The Ant Colony approach is similar to some approaches we reviewed in this section, while the particle swarm approach is a completely new design. We will investigate, amongst other measures, in particular their performance in different scenarios to gain a good understanding of their responsiveness.

## 3 Approaches

### 3.1 Requirements

In designing each load balancing algorithm several important characteristics should be kept in mind. A list of these requirements is provided here:

– *Optimum resource utilization.* A load balancing algorithm should optimize the utilization of resources by optimizing time or cost related to these resources. Since the Grid environment provides a dynamic search space, this optimality is inevitably a partial optimality of the performance.
– *Fairness.* A load balancing algorithm is said to be fair, meaning that the difference between the heaviest loaded node and lightest loaded node in the network is minimized, keeping in mind that the search space is dynamic. The load is defined by the number of jobs assigned to each resource relative to its computational power.
– *Flexibility.* It means that as the topology of the network or the Grid changes, the algorithm should be flexible enough to adhere to the changes in the network.
– *Robustness.* Robustness refers to the fact that when failures in the system occur the algorithm should have a way to deal with the failure and be able to cope with the situation, i.e. not to break down because of a failure. On the contrary, the algorithm should be able to deal with the problem.
– *Distribution.* Distribution for managing resources and running the load balancing algorithm has the benefit of leaving out the single point of failure which centralized approaches are affected by.
– *Simplicity.* By simplicity we try to point out both the size of single software units which are being transferred among resources in the Grid, and also the overhead that these units

bring to resources in order to make load balancing decisions. The size of software units are important as they take up bandwidth when they want to transfer themselves between resources. Since their units are being executed in Grid nodes, there is a preference to keep necessary computations as simple as possible.

### 3.2 Proposed Approaches

In this research, we are suggesting a new approach for applying Ant colony optimization to the problem of load balancing. In the previous approaches, ants act independently from jobs being submitted while in our approach there is a close binding between jobs and load balancing ants. On the other hand, particle swarm has not been used for distributed load balancing in the Grid before and we are proposing a new way to design our load balancing algorithms.

#### *3.2.1 Ant Colony Load Balancing: AntZ*

In this section, a new load balancing algorithm which is developed based on the concepts of ant colony optimization is described. This algorithm (AntZ) is developed by merging the idea of how ants cluster objects with their ability to leave trails on their paths so that it can be a guide for other ants passing their way. We are using the inspiration of how ants are able to cluster objects using an inverse version to spread the jobs in the Grid. In particular, we are making use of the main algorithm proposed by Montresor et. al [18], as well as the adoption of the number of steps before decision making as proposed by Cao [27]. In addition, we add the decay rate from the classical ant approach, as well as we introduce a mutation rate which is specifically added to the problem of load balancing.

A pseudo-code of the AntZ approach is provided in Algorithm 3.1. AntZ is a distributed algorithm and each ant can be considered as an agent working independently. The pseudo-code addresses the main functions that an ant performs during its life cycle. Collectively, all the ants show the desired behaviour by following these steps.

As shown in the pseudo-code, when a job is submitted to a local node in the Grid an ant is initialized and starts working. In each iteration, the ant collects the load information of the node it is visiting (*getNodeLoadInformation()*) and adds it to its history. The ant also updates the load information table of the visited nodes (*localLoadTable.update()*). This load information table of a node contains information of its own load, but also load information of other nodes, which were added to the table when ants visited the node.

When moving to the next node the ant has two choices. One choice is to move to a random node with a probability of mutation rate (*mutRate*). The other choice is to use the load table information in the node to choose where to go. The mutation rate decreases with a *DecayRate* factor as time passes, thus, the ant will be more dependent to load information than to random choice. This iterative process is repeated until the finishing criteria is met which is a predefined number of steps. Finally, the ant delivers its job to the node and finishes its task.

When an ant visits a node, it updates the node's load information table with the information of other nodes, but at the same time collects the information already provided by the table of that node, if information exists. The load information table acts as a pheromone trail an ant leaves while it is moving, in order to guide other ants to choose better paths rather than wandering randomly in the network. Entries of each local table are the nodes that ants have visited on their way to deliver their jobs together with their load information.

Reading the information in the load table in each node and choosing a direction, which is represented as the *chooseNextStep()* procedure in Algorithm 3.1, the ant uses a simple policy. It chooses the lightest loaded node in the table. The corresponding pseudo-code is provided in Algorithm 3.2. As shown in the algorithm, each entry of the load information table is being evaluated and compared to whether the current load of the visited node is smaller than any other node provided in the load information table. The ant then chooses the node with the smallest load, and in case of a tie, the ant chooses one with an equal probability.

Since the number of jobs submitted to the network increases, the ants can take up a huge amount of bandwidth of the network, thus moving ants should be as simple and small-sized as possible. To account for this, instead of carrying the job while the ant is searching for a “light” node, it can simply carry the source node information to which the job was delivered to, including a unique job id of the source node. Thus, whenever an ant reaches its destination the job can be downloaded from the source as necessary.

**Algorithm 3.1:** ANTZALGORITHM($MutRate$, $MaxSteps$, $DecayRate$)

```
$step \leftarrow 1$
$initialize()$
while $step < MaxSteps$
  do
    $currentLoad \leftarrow getNodeLoadInformation()$
    $AntHistory.add(currentLoad)$
    $localLoadTable.update()$
    if $random() < MutRate$
      then $nextNode = RandomlyChosenStep()$
      else $nextNode = chooseNextStep()$
    $MutRate \leftarrow MutRate - DecayRate$
    $step \leftarrow step + 1$
    $moveTo(nextNode)$
$deliverJobToNode()$
```

The algorithm has some parameters which can be set according to the specific scheduling requirements (i.e. size of the network, job specifications, etc.). The effect of these parameters and their values on the performance of the algorithms are investigated. One of the parameters is *MaxSteps* which defines how many steps an ant should be moving around until it delivers the assigned job to a node in the Grid. If the ant wanders too many steps before delivering the job, it causes an increase in the execution time of each job, and hence, it decreases the performance. On the other hand, if the ant gives up too quickly without moving around then the pheromone (load table information) which it leaves behind decreases, and this in turn decreases the performance of the algorithm. In addition, the ant might not have enough time to encounter a good and light node. Thus, all these parameters should be chosen carefully.

**Algorithm 3.2:** CHOOSENEXTSTEP()

```
$bestNode \leftarrow currentNode$
$bestLoad \leftarrow currentLoad$
for $entry \leftarrow 1$ to $n$
  if $entry.load < bestLoad$
    then $bestNode \leftarrow entry.node$
  else if $entry.load = bestLoad$
    if $random.next < probability$
      then $bestNode \leftarrow entry.node$
```

Another effective parameter which influences the performance of the AntZ algorithm is *MutRate*. As the ants are moving and they are using the load table information to decide which way to go, they sometimes randomly choose an arbitrary node in the Grid to move toward to. The probability of choosing their way randomly is controlled by *MutRate*. *MutRate* decreases with the decay rate, *DecayRate*, while the ant is alive and is searching. This parameter, *DecayRate*, can also have an effect on the performance of the AntZ algorithm.

### *3.2.2 Particle Swarm Optimization: ParticleZ*

Particle swarm optimization has roots in two methodologies. Obvious is its relation to swarm intelligence in general, and to bird flocking, fish schooling, and swarming theory in particular. It is also related to evolutionary computation, and has ties to both genetic algorithms (GA) and evolutionary programming. The system is initialized with a population of random solutions and searches for the optimum solution by updating itself through generations. However, unlike GA, particle swarm optimization (in its standard form) has no evolutionary operators such as crossover and mutation. In particle swarm optimization, the potential solutions, called particles, fly through the problem space by following the current optimum particles [8]. Relationships, similarities and differences between particle swarm optimization and GA are briefly reviewed in [13].

In a particle swarm optimization system, multiple candidate solutions coexist and collaborate simultaneously. Each solution candidate, called a *particle*, flies in the problem search space (similar to the search process for food of a bird swarm) looking for the optimal position to land. A particle, as time passes through its quest, adjusts its position, according to its own *experience*, as well as according to the experience of neighboring particles [6]. There are two main characteristics for each particle in a particle swarm optimization algorithm: its position which defines where the particle lies relative to other solutions in the search space; and its velocity, which defines the direction and how fast the particle should move to improve its fitness. As in any evolutionary algorithm, the fitness of a particle is a number representing how close that particle is to the optimum point compared to other particles in the search space.

One of the advantages of the particle swarm optimization technique over other social behavior inspired techniques is its implementation simplicity. As there are very few parameters to adjust in a particle swarm optimization approach, it is simpler than other evolutionary techniques.

Two factors characterize the particle's status in the search space: its position and its velocity. The $m$-dimensional position for the *ith* particle in the *kth* iteration can be denoted as:

$$x_i(k) = (x_{i1}(k), x_{i2}(k), ..., x_{im}(k)).$$

Similarly, the velocity (i.e., distance change) is also an m-dimensional vector, for the *ith* particle in the *kth* iteration and can be described as:

$$v_i(k) = (v_{i1}(k), v_{i2}(k), ..., v_{im}(k)).$$

The particle updating mechanism for a particle can be formulated as in (1) and (2):

$$\begin{aligned} v_{id}^{k+1} &= w * v_{id}^{k} + c_1 * r_1 * [p_b - x_{id}^{k}] \\ &\quad + c_2 * r_2 * [g_b - x_{id}^{k}] \end{aligned} \tag{1}$$

$$x_{id}^{k+1} = x_{id}^{k} + v_{id}^{k+1} \tag{2}$$

In which $v_{id}^k$, called the velocity for particle $i$ in the *kth* iteration, represents the distance to be travelled by this particle from its current position, $x_{id}^k$ represents the particle position in the *kth* iteration, $p_b$ represents its best previous position (i.e. its experience), and $g_b$ represents the best position among all particles in the population. $r_1$ and $r_2$ are two random functions with a range [0,1], having similar or different distributions. $c_1$ and $c_2$ are positive constant parameters called acceleration coefficients (which control the maximum step size of the particle). The inertia weight $w$, is a user specified parameter that controls, together with $c_1$ and $c_2$, the impact of previous historical values of particle velocities on its current velocity. A larger inertia weight pressures towards global exploration (searching new area), while a smaller inertia weight pressures towards fine-tuning the current search area. Suitable selection of the inertia weight and acceleration coefficients can provide a balance between the global and the local search. The random values involved, prevent the optimization from being caught in a local optima. A detailed analysis on the effect of parameter selection on the convergence of particle swarm optimization is provided in [28].

Using the idea of particle swarm optimization, a new approach for balancing the load in the Grid is proposed. In the ParticleZ algorithm, all the nodes in the Grid are considered as a flock or group of swarms and each node in the Grid is a particle in this flock.

Following the analogy from the particle swarm optimization perspective, the position of each node in the flock can be determined by its load. This definition helps as we search in the load search space and try to minimize the load, thus each node in this search space takes a position according to its load. The velocity of each particle and its position can be defined by the load difference the node has, compared to its other neighbor nodes. Since the particles are trying to balance the load, they can move toward each other by the changes they make to their position (i.e. load), this change in each particle's position can be achieved by exchanging jobs between them. The larger their difference is, the faster they will move toward each other, with a larger velocity.

The different phases of the ParticleZ algorithm consist of a job submission, a queuing, a node communication and a job exchange phase. Taking into account that all nodes are exchanging their loads in parallel, and the dynamic nature of the environment, the network reaches a local optimum quickly. Thus, each node submits some jobs to one of its neighbors, which has the minimum load among all. If all its neighbors are busier than the node itself, no job is submitted from the current node.

The pseudo-code describing this scenario can be seen in Algorithm 3.3. This is the pseudo-code of each individual particle (resource), which runs the ParticleZ algorithm. As can be seen, if there are any jobs in the queue waiting to be executed the node tries to submit them to a lighter node in its neighborhood, and hence achieves a fair load distribution among resources.

In exchanging load from a heavier loaded node to a lighter loaded node, attention must be paid not to burden the lighter node, so that it exceeds the load of the second lightest node among neighbours. If this happens, the distribution of the load is not fair, but a load imbalance is created. To tackle this problem, we define a *threshold* variable, which defines how much load exchange can happen between the nodes. It is calculated by subtracting *lightestLoad* from *secondLightest-Load* among neighbours and the load exchange takes place as long as the *velocity* is greater than the *threshold* value.

**Algorithm 3.3:** ParticleZAlgorithm()

```
$sourceLoad \leftarrow currentNodeLoad()$
while $running$
  do if $jobsQueue.size > 0$
  then
    $lightestLoad \leftarrow$
    $chooseBestNeighbour(sourceLoad)$
    $secondLightestLoad \leftarrow$
    $chooseSecondLightestNeighbour(sourceLoad)$
    $velocity \leftarrow sourceLoad - lightestLoad$
    $threshold \leftarrow lightestLoad - secondLightestLoad$
    while $velocity > threshold$
      do
        $submitJobs(velocity, destLoad)$
        $sourceLoad \leftarrow currentNodeLoad()$
        $velocity \leftarrow sourceLoad - lightestLoad$
```

There are some issues related to the particle swarm optimization, which are necessary to be addressed. In the algorithm we propose, a particle only moves toward its best local neighbour, while in the classical particle swarm optimization algorithm particles keep track of their best global solutions so far. The reason we have not included the history of each particle is that we are dealing with a dynamic environment in which the problem being solved is changing all the time as users are submitting new jobs randomly; thus, the global best solution that the particle has seen is most likely not valid anymore in this dynamic environment.

Equation (3) for updating the velocity of each particle, which was introduced in Section 3.2.2, takes the following form in our design of ParticleZ:

$$v_{id}^{k+1} = g_b - x_{id}^k \tag{3}$$

As mentioned earlier, we are dealing with an environment which is changing dynamically (i.e. the search space is changing), thus the use of the past experience of each particle is not useful; therefore, we assign zero to $c_1$ in order to omit the effect of the past history of the particle. Also again, because of the dynamicity of the problem, the previous velocity should not effect our decision, therefore, we assign a value of zero to $w$ as well. On the other hand, we want to use neighbour particles to identify and decide which one is better to share the work load with, and therefore, we have used a value of one for $c_2$.

In (4), the formula for updating a particle's position is shown, which is the same as the one we introduced in Section 3.2.2. As mentioned, the position of a particle ($x_{id}$) is its load value and it changes while the resource submits jobs to its neighbours.

$$x_{id}^{k+1} = x_{id}^k + v_{id}^{k+1} \tag{4}$$

## 4 Experimental Setup and Results

### 4.1 Setup

#### *4.1.1 GridSim Toolkit*

The GridSim toolkit used as the simulation environment is a java-based discrete-event Grid simulation toolkit. The toolkit supports modelling and simulation of heterogeneous Grid resources (time-shared and space-shared), users and application models. It also provides primitives for the creation of application tasks, mapping of tasks to resources, and their management [3].

Given the implementation nature of the proposed algorithms, a time-shared policy was used for the AntZ algorithm, and a space-shared policy was taken for the ParticleZ algorithm.

The GridSim toolkit supports the modelling and simulation of a wide range of heterogeneous resources, such as single or multiprocessor, shared and distributed memory machines like PCs, workstations, SMPs (Symmetric Multiprocessing), and clusters with different capabilities and configurations. It can also be used for the modelling and simulation of application scheduling on various classes of parallel and distributed computing systems such as clusters, Grids, and P2P networks.

The following are the reasons why the GridSim toolkit was chosen to simulate and evaluate our scheduling algorithms [3]:

- It allows modelling of heterogeneous types of resources.
- Resource capability can be defined in the form of MIPS (Million Instructions Per Second) and SPEC (Standard Performance Evaluation Corporation) benchmark.
- Application tasks can be heterogeneous and they can be CPU or I/O intensive.
- There is no limit on the number of application jobs that can be submitted to a resource.
- Network speed between resources can be specified.
- It supports simulation of both static and dynamic schedulers.
- Statistics of all or selected operations can be recorded. These statistics can then be further analyzed using GridSim statistics analysis methods.

#### *4.1.2 System Model*

For experimental purposes we assume that the Grid consists of a set of resources connected via different communication links with different speeds. In general, each resource may contain multiple computing nodes (machines), and each computing node (machine) may have a single or multiple Processing Elements (PEs). The computational power or the speed of each processor is defined by the number of Cycles Per Unit Time (CPUT). It is actually the GridSim framework's ability that provides us with the definition of the computational power of PEs in CPUT.

Generally, each resource may consist of one or several machines and each machine by itself can have one or multiple processing elements. Processors in each computing node can be heterogeneous, thus, they may have different processing power. In our simulations, without loss of generality and to emphasize on the basic ideas of the algorithms, we assume each resource consists of one machine and each machine is equipped with one or several processors (the variations of this random number for experiments are provided later). The processors in the same or different computing nodes have different processing power.

At any one time, a computing node may have background workload associated with it, which will affect the completion time of the Grid jobs assigned. The GridSim toolkit provides us with the ability to define the background workload according to historical and statistical information for each node. As such, each resource has a background load associated which is taken from the average load that the resource has experienced at similar times (such as working days or weekends).

#### *4.1.3 Application Model*

For our application model, we assume that tasks which are submitted to the Grid (or the application which is being run) consists of a set of independent tasks with no required order of execution. The tasks are of different computational sizes, meaning each task requires a different computation time and data transmission time for completion. The tasks can also have different input and output size requirements.

The length of each task is presented in Millions of Instructions (MI). Tasks can be classified into one of two categories: data-intensive and computationally intensive tasks. In this research, we are concerned with computationally intensive tasks as they are more common in today's real world applications and the waste of computational power of resources is, in general, more costly than their memory.

### 4.1.4 Network Topology

In order to account for different network topologies, a random connection graph for the specific number of resources was generated. First, a Minimum Spanning Tree with all the resources is created, then, random links are added to the tree to generate the final topology of the Grid. Thus, we have control on the number of links and the topology of our Grid in different simulations. For example, when a resource tries to find its neighbours it sends a message in order to retrieve a list of its connected resources.

### 4.1.5 Performance Evaluation Criteria

In this section we define our performance evaluation criteria which are used to evaluate the performance of our algorithms. The criteria include makespan and the load balancing level. In addition, two classical algorithms for comparison purposes are discussed and used. For all measurements taken, we have used an average of thirty runs in order to guarantee statistical correctness.

*Makespan* One of the most common measures in evaluating the performance of a load balancing algorithm is measuring the makespan. The makespan is the "total application execution time". The total application execution time is measured from the time the first job is sent to the Grid, until the last job comes out of the Grid. As we generate Gridlets (term defined in GridSim to represent jobs) and topologies randomly, although every simulation yields roughly the same result, each single simulation is different from another one; thus, we have used an average makespan in order to account for realistic conditions.

*Load Balancing Level* For each resource in the Grid, the load related to that resource is dependent on the number of jobs which are assigned to the node $n_{\text{jobs}}$ by the Grid scheduler and the power of its processing elements $p_i$. Equation (5), shows how the resource load $l_r$ is calculated.

$$l_r = \frac{n_{\text{jobs}}}{\sum_{i=1}^{\max} p_i} \tag{5}$$

The total load $l$ can be calculated using (6). According to this equation when the resource load $l_r$ increases, it results in an increase in the load $l$, and a decrease in $l_r$ decreases $l$. The load $l$ is a value between 0 and 1, where 0 identifies that a resource is not busy and 1 represents a resource being busy.

$$l = 1 - \frac{1}{l_r} \tag{6}$$

One of the aims of a load balancing algorithm is to minimize the variations in workloads on all machines. Regarding this, the standard deviation in workload is often taken as the performance measure of a load balancing algorithm. The smaller the standard deviation, the better the load balancing scheme is. By looking at the changes in the standard deviation of the workload with respect to time, it is easier to visualize the effect of load balancing upon the time of the system [7]. Equation (7), shows the standard deviation of the load in the system.

$$d = \sqrt{\frac{\sum_{i=1}^{n} (\bar{l} - l_i)^2}{n}} \tag{7}$$

In the equation, $\bar{l}$ is the average load of the system and $l_i$ is the load of the *ith* resource.

We define the load balancing level (LBL) $b$ of the system to be a measure of how good the load balancing algorithm is. The load balancing level of the system is defined in (8). The most effective load balancing is achieved when $b$ equals to 100% which implies that $d$ should be zero or close to zero.

$$b = (1 - d) * 100\% \tag{8}$$

*Comparison Against Classical Approaches* We have implemented two common classical approaches (Random and State Broadcast Algorithm) in order to evaluate the performance of our algorithms and discuss their benefits over classical ones. Both algorithms were described in Section 2.1.

### 4.2 Results

In order to evaluate the performance of our algorithms, we first investigate the effect of different

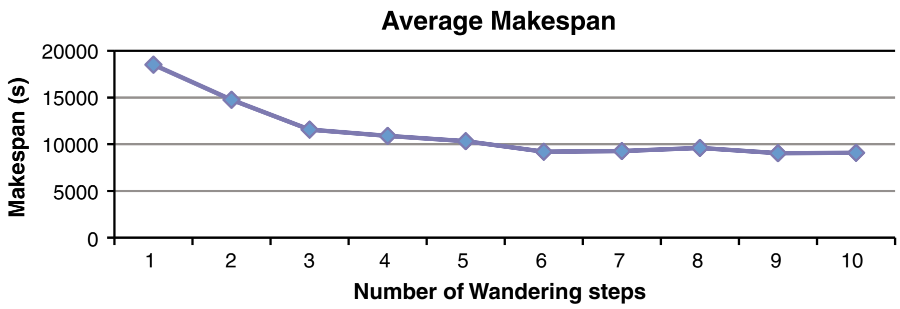


**Fig. 1** Effect of the change in wandering steps on AntZ makespan

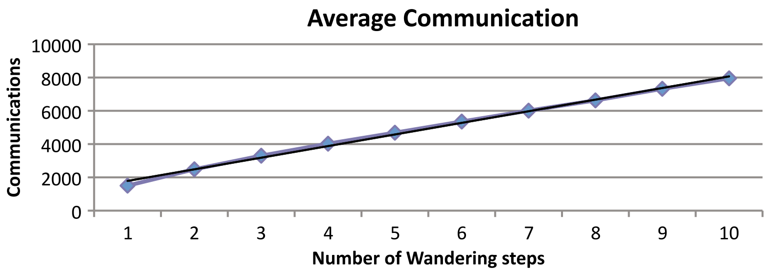


**Fig. 2** Effect of the change in wandering steps on AntZ communication number

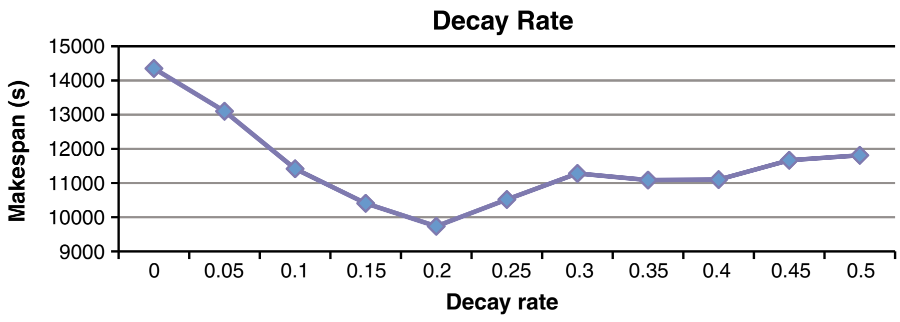


**Fig. 3** Effect of decay rate on AntZ makespan

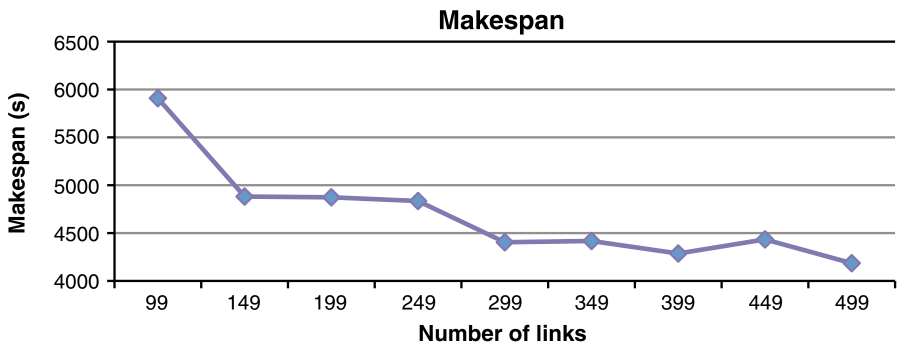


**Fig. 4** Effect of link number on ParticleZ makespan

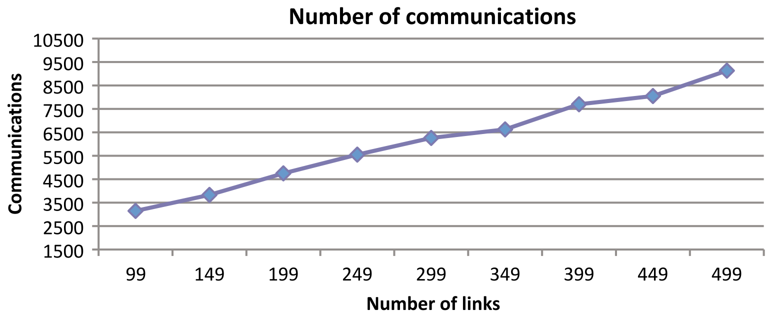


**Fig. 5** Effect of link number on ParticleZ communication number

values for the parameters of each algorithm, and then we investigate a set of experiments to measure the criteria we introduced in the previous section. As described earlier, ParticleZ is implemented with a space-shared FCFS policy inside the resources and the AntZ is accompanied with a time-shared Round-robin policy to schedule the jobs when they are received by a resource. In all the experiments we have compared our algorithms with both the Random and the SBA approach.

### *4.2.1 AntZ Parametric Measurement Effects*

We investigate algorithm-specific performance measures and their effect on the algorithms in the next set of experiments. First, we investigate the effect of wandering steps on the performance of the AntZ algorithm. We have a one hundred node Grid with one thousand jobs being sent to the Grid. Figure 1 shows that as we increase the number of steps an ant wanders until it delivers the job to its destination, the makespan of the algorithm improves, but this increase is larger at the beginning but later on the rate drops to a great extent until it becomes stable.

After about 5 or 6 steps the increase in wandering steps does not seem to have an effect on the performance of the algorithm. The reason behind this phenomenon is that although increasing the number of wandering steps seems to have a positive effect on the performance of the algorithm, as tables are updated more frequently and ants have more time to decide which way to go, but on the other hand, it increases the delay before the jobs are being delivered to the resources, and this delay has a negative effect on the performance.

Figure 2, shows how increasing the number of wandering steps can effect the communication overhead, which is introduced to the system. The figure shows that while we increase the wandering steps, the communication overhead also increases linearly.

In another experiment we measure how different values of the decay rate can effect the performance of the AntZ algorithm. As you remember, while the ant is moving we decrease its mutation rate by a factor; this factor is called the decay rate. By running this experiment we can find out what the best decay rate for a set of specific attributes of a Grid and its jobs is. The results are shown in Fig. 3. For the set of attributes we have, 0.2 is the best decay rate while the mutation rate is set to be 0.5 for this experiment.

### *4.2.2 ParticleZ Parametric Measurement Effects*

In the next set of experiments we measure the effect of different ParticleZ parameter settings on

**Table 1** Grid resource characteristics

| | |
|---|---|
| Number of machines per resource | 1 |
| Number of PEs per machine | 1–5 |
| PE ratings | 10 or 50 MIPS |
| Bandwidth | 1,000 or 5,000 B/S |

**Table 2** Scheduling parameters and their values

| | |
|---|---|
| Number of resources | 100 |
| Number of Gridlets | 1,000 |
| ParticleZ link number | 149 |
| AntZ wander number | 4 |
| AntZ mutation rate | 0.5 |
| AntZ decay rate | 0.2 |

**Table 3** Gridlet characteristics

| | |
|---|---|
| Length | 0–50,000 MI |
| File size | 100 + (10–40%) MB |
| Output size | 250 + (10–50%) MB |

the performance of this algorithm. One of the parameters which can effect the performance of ParticleZ is the number of links that connect resources together. As each particle (resource) communicates with its neighbours to find the lightest node, the number of neighbours can effect the performance of the algorithm.

Figure 4 shows the effect of increasing the number of links and the connectivity of the resources on ParticleZ's makespan. Although it is better to communicate with more resources before exchanging jobs, however, it is not always good as communication with more resources adds an extra time overhead, which prevents a significant improvement in the performance of the system.

Figure 5 shows the effect of increasing the number of links on the communication overhead of the ParticleZ algorithm. As can be seen in the figure, it has a linear growth with an increasing number of links.

### *4.2.3 Makespan and Simulation Time*

The characteristics of the resources as Grid resources we have used for all of the following experiments are shown in Table 1. There is one machine for each Grid resource and each machine has a random number of PEs ranging between 1 and 5. Each PE has a different processing power. Without loss of generality, we set the local load factor for resources to be zero; this does not effect the performance measure of the algorithms. Setting it to zero helps us analyze the effect and behaviour of the algorithms better.

For the next set of experiments, we compare the makespan of the different algorithms. The values for the different parameters of each algorithm are shown in Table 2.

As said earlier, the Gridlets which are sent to the Grid are supposed to be independent of each other. The characteristics of the Gridlets sent to the Grid to compare the makespan of different algorithms are shown in Table 3.

Figure 6 shows a comparison between the makespan of the different algorithms with parameter specifications described above. The experimental results show SBA is performing best amongst all. This is expected as the SBA is keeping track of the state of all the resources at each point in time, which enables it to make optimal decisions at each point in time than all the other approaches. After SBA, ParticleZ wins the competition by having the second smallest makespan. Comparing ParticleZ and AntZ with each other, ParticleZ performs better than AntZ by a factor of 1.72; also, ParticleZ performs better than Random-SpaceShared by a factor of 3.42, and AntZ performs better than Random-TimeShared by a factor of 1.83.

One important but hidden drawback that SBA is subjected to is related to the overall cpu cycles and the time it takes for the Grid to execute it. Since there is a copy of the system state vector in all machines, each machine is using some time and cpu cycles to search the state vector individually in order to schedule each task. This causes a lot of cpu cycles to be wasted, but as we are running a parallel platform this disadvantage cannot be

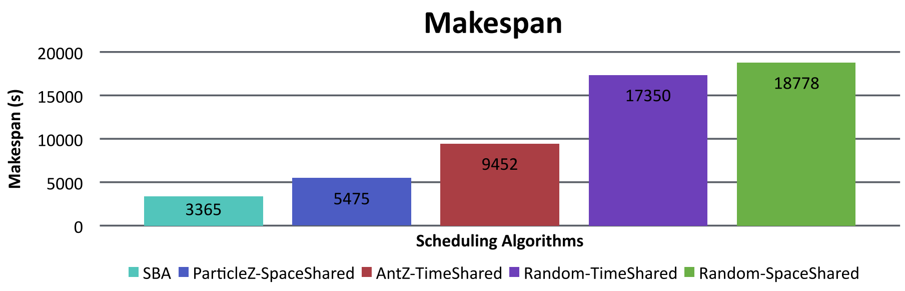


**Fig. 6** Comparing the makespan of different approaches

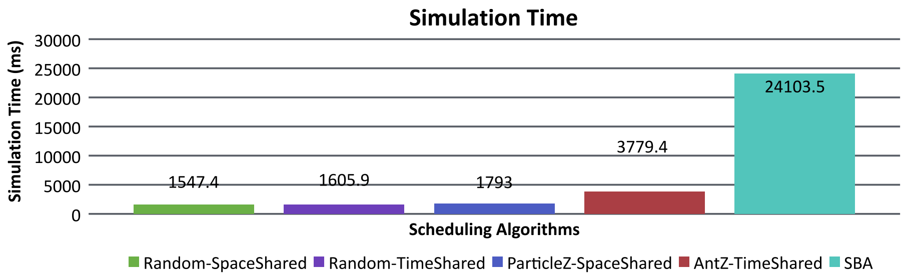


**Fig. 7** Simulation time related to each algorithm in milliseconds

observed. This negative effect is shown in Fig. 7 and the time shown in this figure shows only the simulation time of each algorithm. As the simulation is measured on a single machine, the effect of parallelism is discarded and SBA, although having a very low makespan, actually takes longer to run and it is because all these wasted seconds can not be seen in the previous figure because of parallelism. This effect is even worse as the number of resources grow in the Grid. Please note that this figure only shows the simulation time and does not count for different job lengths, etc.

Another drawback related to SBA is the number of communications it takes. Figure 8 shows the number of extra communications of each algorithm to achieve the load balancing. For ParticleZ, each communication message a node sends to its neighbours to acquire their load status and its response, and each job exchange between two resources is considered as communication. For AntZ, each ant taking a step while searching for the best node to deliver the job to, is considered as a communication. Finally for SBA, each broadcast message a resource sends to other resources is considered as communication overhead. The numbers shown in the figure are the average values of thirty runs with the same parameter settings as described earlier. As shown in the figure, AntZ has a higher communication overhead compared to ParticleZ. Obviously, the other two random approaches have no communication overhead at all, thus, they are not shown in the figure. SBA has the highest number of communications by a factor of around 1300. This large number of communications can be a bottleneck for the network and in scenarios with congested networks the probability of messages being lost increases.

### *4.2.4 Scalability*

In the next experiment we investigate how fair each of the algorithms is. Table 4 shows the load balancing level of the system described earlier in (8) along with their standard deviation from several runs. The closer the value approaches 100%, the better the load balancing of the algorithms is. It means that the load is spread more fairly among all the resources. According to the experimental results both, ParticleZ and SBA, have the best load balancing levels. AntZ along with the other random approaches ranks third in spreading the load uniformly among resources.

As can be seen in Fig. 9, all the algorithms show a linear growth in response to the increasing number of jobs. However, SBA along with the proposed approaches show a much slower growth compared to the random approaches. Among

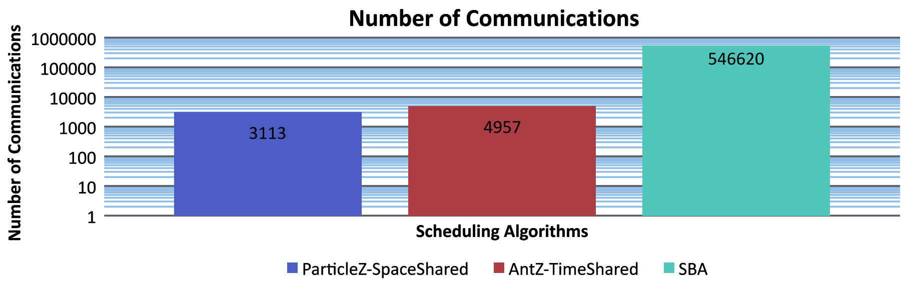


**Fig. 8** Communication overhead related to each algorithm

**Table 4** Average load balancing level of the system for different algorithms

| Algorithm | Average LBL (%) |
|---|---|
| ParticleZ-SpaceShared | 81±2.09 |
| SBA | 80±0.47 |
| Random-SpaceShared | 67±1.29 |
| AntZ-TimeShared | 65±0.70 |
| Random-TimeShared | 62±0.95 |

**Table 5** Predicting execution time based on number of jobs

| Algorithm | Prediction trend line (in seconds) |
|---|---|
| SBA | $762.5 * n_{\text{jobs}} + 808.5$ |
| ParticleZ-SpaceShared | $906.7 * n_{\text{jobs}} + 1782$ |
| AntZ-TimeShared | $2{,}478 * n_{\text{jobs}} + 1{,}291$ |
| Random-TimeShared | $5{,}518 * n_{\text{jobs}} - 291.2$ |
| Random-SpaceShared | $6{,}069 * n_{\text{jobs}} - 1{,}419$ |

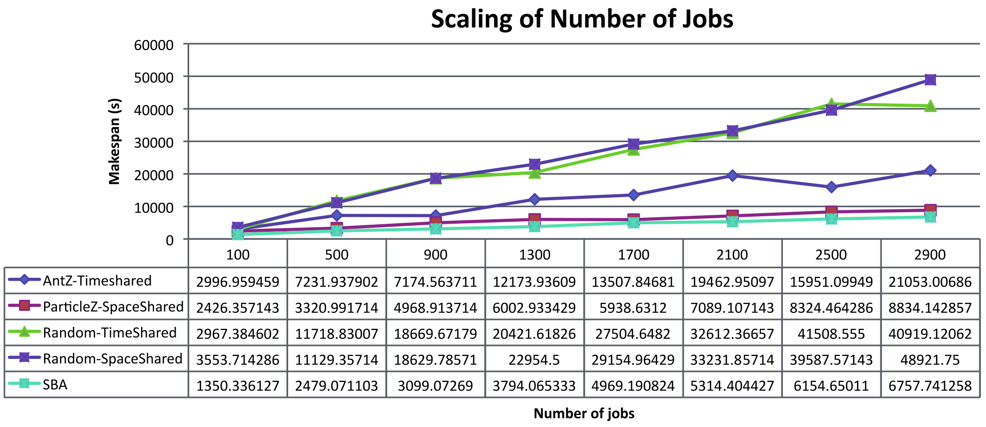
them, ParticleZ and SBA are quite close to each other. Table 5 shows each algorithm with its prediction trend line for a 100 node Grid. As can be seen, ParticleZ and SBA have smaller slopes among all other approaches.

In the next set of experiments we investigate the effect of increasing the number of jobs on the performance of the algorithms. Thus, we keep a fixed number of resources and run the experiments while we increase the number of jobs being sent to the Grid. The specifications and parameter settings of the algorithms and the system are listed in Tables 1–3.

In Fig. 10, we investigate the effect of increasing the length of jobs on the performance of the algorithms. Length of the jobs is defined in Millions of Instructions (MIs) in GridSim. Parameter settings to run this experiment are the same as described in Tables 1–3. We increase the length of the Gridlets by adding 250,000 MIs at each step and investigate its effect on the makespan. The numbers at the bottom of the figure show the execution time for each algorithm. As can be seen, the growth is linear for all the approaches and the results show the best performance is achieved by both the ParticleZ and the SBA algorithm. AntZ ranks third and the other two random approaches, as expected, do not respond well to the larger lengths of Gridlets, but for small Gridlet lengths they can perform comparably to others.

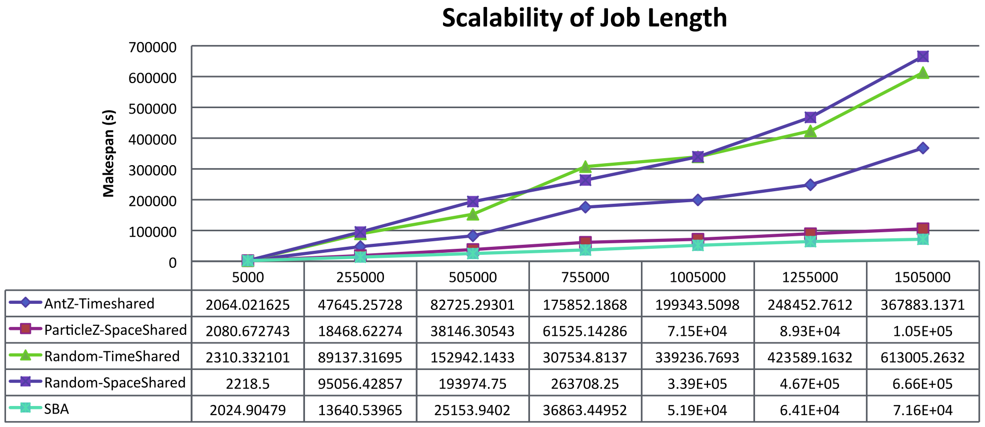
### 4.2.5 Resource Effect and Injection Points

Figure 11 shows how increasing the number of resources, while keeping the same number of jobs being sent to the Grid constant, decreases the performance of the Grid in terms of an increase in execution time. In this experiment, 3,000 jobs are sent to the Grid with varying number of resources, and as can be seen increasing the number of resources has a decreasing effect on the execution time. ParticleZ and SBA are performing better when we have a small number of resources (50) and a large number of jobs compared to the number of resources (3,000). As the number of resources increases the per-



| | 100 | 500 | 900 | 1300 | 1700 | 2100 | 2500 | 2900 |
|---|---|---|---|---|---|---|---|---|
| ◆ AntZ-Timeshared | 2996.959459 | 7231.937902 | 7174.563711 | 12173.93609 | 13507.84681 | 19462.95097 | 15951.09949 | 21053.00686 |
| ■ ParticleZ-SpaceShared | 2426.357143 | 3320.991714 | 4968.913714 | 6002.933429 | 5938.6312 | 7089.107143 | 8324.464286 | 8834.142857 |
| ▲ Random-TimeShared | 2967.384602 | 11718.83007 | 18669.67179 | 20421.61826 | 27504.6482 | 32612.36657 | 41508.555 | 40919.12062 |
| ■ Random-SpaceShared | 3553.714286 | 11129.35714 | 18629.78571 | 22954.5 | 29154.96429 | 33231.85714 | 39587.57143 | 48921.75 |
| ■ SBA | 1350.336127 | 2479.071103 | 3099.07269 | 3794.065333 | 4969.190824 | 5314.404427 | 6154.65011 | 6757.741258 |

**Fig. 9** Effect of the increase in number of jobs on performance of the algorithms



| | 5000 | 255000 | 505000 | 755000 | 1005000 | 1255000 | 1505000 |
|---|---|---|---|---|---|---|---|
| AntZ-Timeshared | 2064.021625 | 47645.25728 | 82725.29301 | 175852.1868 | 199343.5098 | 248452.7612 | 367883.1371 |
| ParticleZ-SpaceShared | 2080.672743 | 18468.62274 | 38146.30543 | 61525.14286 | 7.15E+04 | 8.93E+04 | 1.05E+05 |
| Random-TimeShared | 2310.332101 | 89137.31695 | 152942.1433 | 307534.8137 | 339236.7693 | 423589.1632 | 613005.2632 |
| Random-SpaceShared | 2218.5 | 95056.42857 | 193974.75 | 263708.25 | 3.39E+05 | 4.67E+05 | 6.66E+05 |
| SBA | 2024.90479 | 13640.53965 | 25153.9402 | 36863.44952 | 5.19E+04 | 6.41E+04 | 7.16E+04 |

**Fig. 10** Effect of the increase in job length on performance of the algorithms

formance, the difference between the algorithms decreases.

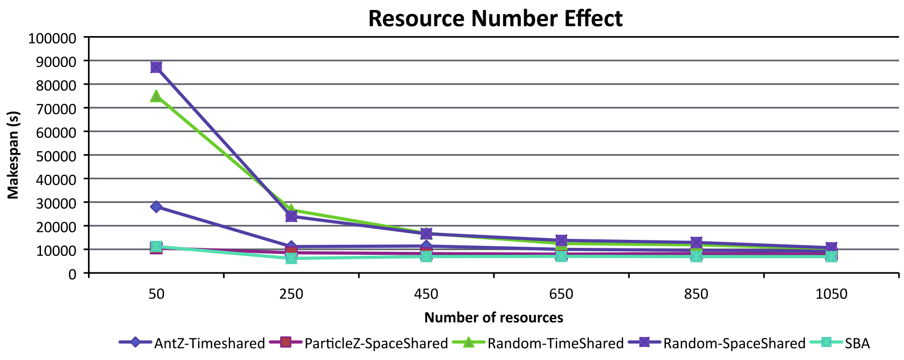
One of the very interesting performance questions which arises in a distributed algorithm like AntZ and ParticleZ is: how do the algorithms respond if all the jobs are injected from a single point in the Grid? From the AntZ's perspective it will take longer to build the load table information, and from the ParticleZ's perspective it will have a negative effect as the jobs will need more time to be spread evenly. We have investigated this effect to see how much it will slow down or have a negative effect on the performance of the algorithms.

The random approaches obviously perform very poorly if we send all jobs to one node. Figure 12 shows AntZ copes better than ParticleZ in response to all the jobs being sent to one node in the Grid. The reason lies in the mutation factor which is incorporated inside AntZ. With the mutation, an ant moves randomly from time to time which helps to build up the load tables more quickly to overcome the negative effect. It can be inferred from the figure, that ParticleZ's performance decreases by a factor of 2.4 for a one hundred node network with Gridlets of a length between 0 and 50,000. On the other hand, AntZ's performance decreases by a factor of 1.36 in the same scenario setting.

### *4.2.6 Resource Heterogenity*

In analyzing the performance of the introduced algorithms one characteristic that is of importance in real world scenarios is how each algorithm responds to different heterogeneity of jobs and resources. Figure 13 shows a comparison of different makespans for both high and low resource heterogeneity as well as high and low job heterogeneity. In this analysis, high resource het-

**Fig. 11** Effect of increasing number of resources on execution time



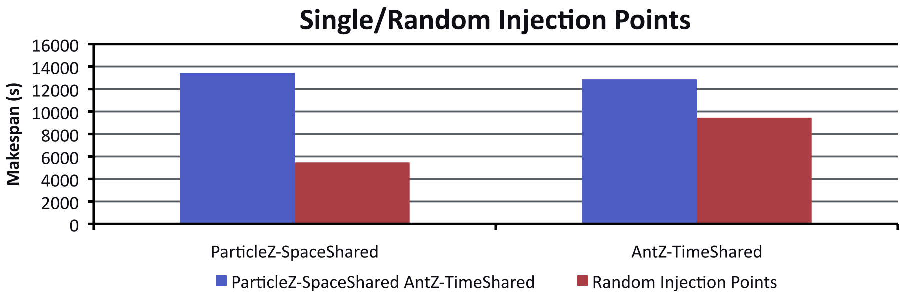


**Fig. 12** Effect of single and random injection points on the performance of the algorithms

erogeneity is simulated by each resource having a random number of PEs between 6 and 20. The low resource heterogeneity is defined by having the number of PEs between 1 and 5. High job heterogeneity is simulated by having the length of the Gridlets between 0 and 500,000 MIs, while low job heterogeneity refers to the condition whereby Gridlet lengths are between 40,000 to 50,000 MIs.

By analyzing the results in Fig. 13 we can see when both resource heterogeneity and job heterogeneity are low, ParticleZ outperforms all the other approaches. SBA performs very close to ParticleZ, while the AntZ approach ranks third.

We can see a similar trend in high job heterogeneity with low resource heterogeneity. When having a high resource heterogeneity, SBA outperforms all other approaches with both high and low job heterogeneity, however, ParticleZ is very close to SBA in terms of performance. The other three approaches have higher execution times.

## 4.3 Summary and Discussion

We investigated several algorithm-related parametric effects for both the AntZ and ParticleZ algorithms. We investigated the effect of different wandering steps on the execution time and the communication overhead of the AntZ algorithm. The results show as we increase the number of wandering steps the performance of the AntZ improves, but there is a limit to this improvement after which the performance remains the same, although the number of wandering steps increases. We also studied the effect of different decay rates on the performance of the AntZ, and identified the best decay rate for our simulation setting.

For the ParticleZ algorithm, we investigated the effect of different link numbers on both the execution time and the communication overhead of the algorithm. The communication overhead grows linearly by increasing the number of links while the makespan decreases.

Analyzing the results of the makespan shows that SBA has the smallest makespan among all and ParticleZ performs better than AntZ in this regard. Although SBA has the smallest makespan among all the approaches, comparing its simulation time with others reveals that there are many computational activities going on in parallel in all machines to execute SBA, which although it does not effect the overall makespan, it increases the

**Fig. 13** Effect of heterogeneity of jobs and resources on the makespan (measured in seconds)

| Algorithms | Resource Heterogeneity | | | |
|---|---|---|---|---|
| | Low | | High | |
| | Job Heterogeneity | | | |
| | Low | High | Low | High |
| SBA | 18524 | 190941 | 8136 | 94983 |
| ParticleZ-SpaceShared | 16372 | 153338 | 10606 | 103756 |
| AntZ-TimeShared | 45391 | 475181 | 24829 | 332725 |
| Random-TimeShared | 66074 | 774329 | 56560 | 541741 |
| Random-SpaceShared | 62457 | 750594 | 51194 | 603408 |

computational complexity for the overall Grid, and therefore, makes SBA the worst approach among all in terms of computational complexity.

Comparing the number of communications each algorithm is concerned with, SBA shows the largest growth in communication cost compared to the other two approaches, whereby ParticleZ involves the smallest number of communications among all.

ParticleZ also wins the competition among all other approaches regarding the “fairness” measure, as it has the highest load balancing level amongst all other approaches.

Looking at the scalability of the algorithms, all approaches show a linear growth in response to an increase in the number of jobs. ParticleZ and SBA have the smallest gradient and are very close to each other, AntZ ranks third.

Regarding an increase in the length of jobs, all approaches show a linear increase; however, ParticleZ along with SBA are best among all. Furthermore, an increase in the number of resources decreases the makespan.

Overall, ParticleZ proves to perform slightly better than AntZ in many regards. On the other hand, looking at the results it shows that ParticleZ has most of the advantages of SBA without having its disadvantages. However, there is one drawback associated with ParticleZ. When jobs are sent to the Grid and are submitted to one or a small number of resources and are not spread throughout the Grid, ParticleZ’s performance decreases more than AntZ’s performance. The reason is the mutation factor incorporated within AntZ, which makes it better to deal with such situations.

## 5 Conclusion

In this research we have investigated the use of swarm intelligence inspired techniques in designing distributed Grid load balancing algorithms. Specifically, we have taken inspiration from social insect systems and sociological behaviour of birds and school of fishes.

The contributions of the designed algorithms (AntZ and ParticleZ) can be categorized as follows: (1) Many centralized load balancing approaches have been developed and applied to the Grid, even those with inspiration taken from swarm intelligence techniques such as genetic algorithms and Tabu search, however, the centralized approaches have many drawbacks as we previously outlined. Research using swarm intelligence techniques for distributed load balancing has only started to be investigated. (2) Although a variety of ant colony inspired approaches have been used for distributed load balancing, there is no comparison of these approaches with any other distributed swarm intelligence technique. In this research, we compared the performance of the ant colony approach with another swarm intelligence technique, particle swarm optimization. We performed measurements to compare the two algorithms in order to identify which are more effective and under which conditions. We also compared the performance of the algorithms with other classical techniques. (3) Particle swarm optimization has been used to address the problem of centralized load balancing [1, 6, 21], but it has never been used for distributed load balancing in a dynamic environment such as the Grid. (4) Most of the research and experimental results, especially in the area of distributed load balancing and ant colony, have used their own developed infrastructure to simulate the performance of their approaches, thus, the question remains how well they would perform in a real world environment. We have used a real world simulation platform, *GridSim*, which provides us with a reliable toolkit by allowing evaluations to be done under realistic conditions.

This research investigated two different approaches (inspired by ant colony and particle swarm optimization) for developing load balancing algorithms and it shows the benefits of swarm intelligence techniques in the distributed Grid load balancing domain. Furthermore, we showed, although particle swarm optimization has not been used widely in designing distributed load balancing algorithms, it performs quite well and it even outperforms the ant colony approach in many scenarios. One of the important characteristics of the designed algorithms compared to central approaches is their responsiveness to scalability of the Grid. In centralized approaches, an increase in the number of resources in the Grid can always be a problem as the information of all the resources has to be stored and must be recalled at all time, but our distributed approaches work quite well with a large number of resources and jobs. This drawback was seen by running the SBA simulations with a large number of resources and examining the simulation time.

The advantages of our proposed algorithms can be summarized as follows: 1) Looking at the simulation results, the algorithms show good performance results and optimized resource utilization. 2) The algorithms have proved to be “fair” compared to a random and SBA approach. ParticleZ has a load balancing level of 81%, SBA has a load balancing level of 80%, AntZ achieves a load balancing level of 65% and the random approaches have a load balancing level close to 65%. 3) Both ParticleZ and AntZ are flexible approaches in dealing with the changes that happen in the Grid. 4) Both proposed approaches are distributed in nature. As the algorithms have taken inspiration from sociological systems, being distributed is an inherent part and we used this ability in designing the approaches. 5) Both algorithms are very simple which is a benefit for a distributed system. In the AntZ approach, the ants which have to move among resources to find the best resource to deliver the job to, are very small in size and perform small computations in each resource. The ParticleZ algorithm implements simple computations as it only sends small messages and has to choose the lightest resource amongst all neighbour resources. 6) Looking at the scalability of the algorithms they show linear growth in response to both an increase in the number of jobs and an increase in the length of jobs.

In conclusion, we can say classical approaches such as Random and SBA, although suitable for small sized networks, are not efficient for large Grids.

## 6 Future Work

The algorithms in their current state do not address the problem of dynamic resource failure in the Grid. A mechanism should be in place that prevents Gridlet loss while any resource in the Grid shuts down. Another issue which is worth addressing is the special scenario when all resources in the Grid are too busy to take new jobs on. The question arises what should be done with new jobs that are submitted to the Grid. Another issue worth investigating is that although we have simulated the algorithms within a simulation framework similar to a real world scenario; it may still need some small modification. For example, we have not considered issues related to security in this research. One of the steps which can be taken toward adding security is limiting ants from performing particular actions in different resources.

One of the important issues in large-scale Grids and peer-to-peer systems is resource failures and the robustness of the system. As the size of the Grids is continually increasing, the probability of resource failures also increases. As such, developing fault tolerant algorithms which are able to deal with these failures are gaining more and more attention. Failures which happen in a Grid environment can be divided into two categories.

A resource may shutdown manually, thus, it can send a notice message or perform some additional steps before shutting down. Or the resources may fail suddenly without any notice. Thus, we need to incorporate a mechanism to deal with both cases of resource failures in our system without effecting jobs submitted by users.

The ParticleZ algorithm can deal with failures more easily. At the time a resource wants to share its workload with other resources, it simply sends a message and queries about its available neighbours, therefore, whenever a resource breaks down it is automatically eliminated from this process. Thus, in case of ParticleZ, a message sent to the user reporting on the uncompleted jobs would suffice.

However, when a resource fails without further notice the situation is more complex. One possible solution is the following. When a job is sent to the Grid by a user, the worst case execution time will be estimated for that job. This predicted time represents the worst case in which the user must have received the results of its job submission. An event is then scheduled for the predicted time. At this specific time, the user will check whether the job result was returned; if the job result has already come back successfully, no further actions will be taken; otherwise, the job will be resubmitted and the whole process repeats.

Another important issue is to pay attention that the nodes may not be dedicated nodes in the Grid, and each may have their own background workload. Thus, the local load of each resource should be incorporated in the load calculation equation, which effects the decision making of the algorithms accordingly.

In this research, we have simulated the proposed algorithms with a simulation platform developed for the Grid, and the results proved to be promising. The next step would be to apply the algorithms in a real world Grid or incorporate the algorithm in existing Grid toolkits such as the Sun Grid Engine or Globus toolkit to confirm the simulation results.

## References

1. Abraham, A., Liu, H., Zhang, W., TChang, G.: Scheduling jobs on computational Grids using fuzzy particle swarm algorithm. In: Proceedings of 10th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, pp. 500–507 (2006)
2. Al-Dahoud, A., Belal, M.: Multiple ant colonies for load balancing in distributed systems. IN: Proceedings of The first International Conference on ICT and Accessibility (2007)
3. Buyya, R., Murshed, M.M.: Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. Concurr. Comput.: Pract. Exp. **14**, 1175–1220 (2002)
4. Cao, J.: Self-organizing agents for Grid load balancing. In: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, pp. 388–395 (2004)
5. Di Caro, G., Dorigo, M.: Antnet: distributed stigmergetic control for communications networks. J. Artif. Intell. Res. **9**, 317–365 (1998)
6. Chen, T., Zhang, B., Hao, X., Dai, Y.: Task scheduling in Grid based on particle swarm optimization. In: ISPDC '06: Proceedings of the Fifth International Symposium on Parallel and Distributed Computing, pp. 238–245. IEEE Computer Society, Washington (2006)
7. Chow, K.P., Kwok, Y.K.: On load balancing for distributed multiagent computing. IEEE Trans. Parallel Distrib. Syst. **13**(8), 787–801 (2002)
8. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, MHS '95., pp. 39–43 (1995)
9. Foster, I., Kesselman, C.: The Grid in a Nutshell. In: Nabrzyski, J., Schopf, J.M. (eds.) Grid Resource Management: State of the Art and Future Trends, pp. 3–13. Kluwer Academic Publisher, Boston (2004)
10. Grosan, C., Abraham, A., Helvik, B.: Multiobjective evolutionary algorithms for scheduling jobs on computational Grids. In: Guimares, N., Isaias, P. (eds.) ADIS International Conference, Applied Computing. Salamanca, Spain (2007)
11. Heusse, M., Guerin, S., Snyers, D., Kuntz, P.: A new distributed and adaptive approach to routing and load balancing in dynamic communication networks, 4 pp. Technical Report (1998)
12. Kandagatla, C.: Survey and taxonomy of grid resource management system. http://www.cs.utexas.edu/users/browne/cs395f2003/projects/KandagatlaReport.pdf (2003)
13. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948 (1995)
14. Kwang, M.S., Sun, H.W.: Ant colony optimization for routing and load-balancing: survey and new directions. IEEE Trans. Syst. Man Cybern. Part A **33**(5), 560–572 (2003)
15. Li, Y., Lan, Z.: A Survey of Load Balancing in Grid Computing. Springer, Berlin (2005)
16. Livny, M., Melman, M.: Load balancing in homogeneous broadcast distributed systems. SIGMETRICS Perform. Eval. Rev. **11**(1), 47–55 (1981)
17. Moallem, A., Ludwig, S.A.: Using techniques for distributed Grid job scheduling. In: Proceedings of the 24th Annual ACM Symposium on Applied Computing (2009)
18. Montresor, A., Meling, H., Babaoglu, O.: Messor: Load-Balancing through a Swarm of Autonomous Agents. Springer, Berlin
19. Salehi, M.A., Deldari, H.: Grid load balancing using an echo system of intelligent ants. In: PDCN'06: Proceedings of the 24th IASTED International Conference on Parallel and Distributed Computing and Networks, vol.52, 47 pp. ACTA Press, Anaheim, CA, USA (2006)
20. Salleh, S., Zomaya, A.Y.: Scheduling in Parallel Computing Systems: Fuzzy and Annealing Techniques. The Springer International Series in Engineering and Computer science (1999)
21. Salman, A., Ahmad, I., Al-Madani, S.: Particle swarm optimization for task assignment problem. Microprocess. Microsyst. **26**, 363–371 (2002)
22. Schoonderwoerd, R., Holland, O., Bruten, J.: Ant-like agents for load balancing in telecommunications networks. In: AGENTS '97: Proceedings of the first international conference on Autonomous agents, pp. 209–216. ACM, New York (1997)
23. Schoonderwoerd, R., Holland, O.E., Bruten, J.L., Rothkrantz, L.J.M.: Ant-based load balancing in telecommunications networks. Adapt. Behav. **2**, 169–207 (1996)
24. Shivaratri, N.G., Krueger, Ph., Singhal, M.: Load distributing for locally distributed systems. Computer **25**(12), 33–44 (1992)

25. Sim, K.M., Sun, W.H.: Multiple Ant Colony Optimization for Load Balancing. Springer, Berlin (2003)
26. Subrata, R., Zomaya, A.Y.: A comparison of three artificial life techniques for reporting cell planning in mobile computing. IEEE Trans. Parallel Distrib. Syst. **14**(2), 142–153 (2003)
27. Subrata, R., Zomaya, A.Y., Landfeldt, B.: Artificial life techniques for load balancing in computational Grids. J. Comput. Syst. Sci. **73**(8), 1176–1190 (2007)
28. Trelea, I.C.: The particle swarm optimization algorithm: convergence analysis and parameter selection. Inf. Process. Lett. **85**(6), 317–325 (2003)
29. White, T., Pagurek, B.: Asga: improving the ant system by integration with genetic algorithms. In: University of Wisconsin, pp. 610–617. Morgan Kaufmann (1998)
30. Yagoubi, B., Slimani, Y.: Dynamic load balancing strategy for Grid computing. in: Proceedings of World Academy of Science, Engineering and Technology (2006)
31. Zhu, W., Sun, Ch., Shieh, C.: Comparing the performance differences between centralized load balancing methods. IEEE International Conference on Systems, Man, and Cybernetics, vol. 3, pp. 1830–1835 (1996)
32. Derbal, Y.: Entropic Grid scheduling. J Grid Computing **4**, 373–394 (2006)
33. Ranganathan, K., Foster, I.: Simulation studies of computation and data scheduling algorithms for data Grids. J Grid Computing **1**, 53–62 (2003)
34. Lucchese, F., Huerta, E.J., Yero, Sambatti, F.S.: An adaptive scheduler for Grids. J Grid Computing **4**, 1–17 (2006)