Handbook of Research on Computational Intelligence for Engineering, Science, and Business

Siddhartha Bhattacharyya
RCC Institute of Information Technology, India

Paramartha Dutta
Visva Bharati University, India

Volume I

Information Science
REFERENCE

Managing Director: Lindsay Johnston
Editorial Director: Joel Gamon
Book Production Manager: Jennifer Romanchak
Publishing Systems Analyst: Adrienne Freeland
Development Editor: Austin DeMarco
Assistant Acquisitions Editor: Kayla Wolfe
Typesetter: Lisandro Gonzalez
Cover Design: Nick Newcomer

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Handbook of research on computational intelligence for engineering, science, and business / Siddhartha Bhattacharyya and Paramartha Dutta, editors.
pages cm
Includes bibliographical references and index.
Summary: “This book discusses the computation intelligence approaches, initiatives and applications in the engineering, science and business fields, highlighting that computational intelligence as no longer limited to computing-related disciplines and can be applied to any effort which handles complex and meaningful information”-- Provided by publisher.
ISBN 978-1-4666-2518-1 (hardcover) -- ISBN (invalid) 978-1-4666-2519-8 (ebook) -- ISBN (invalid) 978-1-4666-2520-4 (print & perpetual access) 1. Computational intelligence. 2. Content analysis (Communication) I. Bhattacharyya, Siddhartha, 1975- II. Dutta, Paramartha.
Q342.H36 2013
006.3--dc23

2012027413

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 16
Schedulers Based on Ant Colony Optimization for Parameter Sweep Experiments in Distributed Environments

Elina Pacini
Institute for Information and Communication Technologies, Universidad Nacional de Cuyo, Argentina

Cristian Mateos
Instituto Superior de Ingenieria de Software, Consejo Nacional de Investigaciones Cientificas y Tecnicas, Argentina

Carlos García Garino
Institute for Information and Communication Technologies, Universidad Nacional de Cuyo, Argentina

ABSTRACT

Scientists and engineers are more and more faced to the need of computational power to satisfy the ever-increasing resource intensive nature of their experiments. An example of these experiments is Parameter Sweep Experiments (PSE). PSEs involve many independent jobs, since the experiments are executed under multiple initial configurations (input parameter values) several times. In recent years, technologies such as Grid Computing and Cloud Computing have been used for running such experiments. However, for PSEs to be executed efficiently, it is necessary to develop effective scheduling strategies to allocate jobs to machines and reduce the associated processing times. Broadly, the job scheduling problem is known to be NP-complete, and thus many variants based on approximation techniques have been developed. In this work, the authors conducted a survey of different scheduling algorithms based on Swarm Intelligence (SI), and more precisely Ant Colony Optimization (ACO), which is the most popular SI technique, to solve the problem of job scheduling with PSEs on different distributed computing environments.

DOI: 10.4018/978-1-4666-2518-1.ch016

INTRODUCTION

Parameter Sweep Experiments, or PSEs for short, is a very popular way of conducting simulation-based experiments among scientists and engineers through which the same application code is run several times with different input parameters resulting in different outputs (Youn and Kaiser, 2010). Representative examples of PSEs are sensitivity studies of results in terms of defined parameter changes like is the case of imperfections in the simulation of simple tension test, or the study of buckling of imperfect columns.

From a purely software perspective, most PSEs are cluster friendly in the sense that individual inputs of an experiment can be handled by independent jobs. Therefore, using a software platform such as Condor (Thain et al., 2005), which is able to exploit the distributed nature of a computer cluster, allows these jobs to be run in parallel. In this way, not only PSEs execute faster, but also experiments more computing intensive can be computed, and hence more complex simulations can be performed. The same idea has been systematically applied to execute PSEs on Grids (Foster and Kesselman, 2003), which are basically infrastructures that connect clusters via wide-area connections to increase computational power. To this end, software platforms designed to exploit Grids provide the illusion of the existence of a large supercomputer, which in turn virtualizes and combines the hardware capabilities of many much less powerful, geographically-dispersed machines to run resource intensive applications (Coveney et al., 2005).

On the downside, for users not proficient in distributed technologies, manually configuring PSEs is tedious, time-consuming and error-prone. As a consequence, users typically waste precious time that could be instead invested into analyzing results. The availability of elaborated GUIs -especially for Grids- that help in automating an experimentation process has in part mitigated this problem. However, the highly complex nature of today's experiments and thus their associated computational cost greatly surpasses the time savings that can be delivered by this automation.

A recent distributed computing paradigm that is rapidly gaining momentum is Cloud Computing (Buyya et al., 2009), which bases on the idea of providing an on demand computing infrastructure to end users. Typically, users exploit Clouds by requesting from them one or more machine images, which are virtual machines running a desired operating system on top of several physical machines (e.g. a datacenter). Interaction with a Cloud is performed by using *Cloud services*, which define the functional capabilities of a Cloud, i.e. machine image management, access to software/data, security, and so on.

Due to the fact that PSEs perform the processing of a lot of jobs, it is necessary to address how they will be executed in distributed computing environments, which is a complex endeavor. For jobs to be properly executed, it is necessary to allocate them to different resources reasonably and optimally. This problem is known as job scheduling and is an NP-complete problem, which aims to minimize the overall execution time of all jobs. Job scheduling is one of the bottlenecks in distributed computing.

In the last ten years or so, Swarm Intelligence has received increasing attention in the research community. Swarm Intelligence refers to the collective behavior that emerges from a swarm of social insects (Bonabeau et al., 1999). Social insect colonies solve complex problems collectively by intelligent methods. These problems are beyond the capabilities of each individual insect, and the cooperation among them is largely self-organized without any supervision. Through studying social insect colonies behaviors such as ant colonies, researchers have proposed some algorithms or theories for combinational optimal problems. Moreover, job scheduling in Grids or Clouds is also a combinational optimal problem.

Motivated by these facts, we conducted a literature review of job scheduling techniques based on Ant Colony Optimization (Dorigo, 1992) algorithms. The inspiring source of ACO is the foraging behavior of real ants. ACO is one of the most popular optimization techniques in the area of Swarm Intelligence (García Martínez et al., 2007; Huang and Wang, 2011; Chandra and Baskaran, 2012) and the most popular optimization technique among bioinspired techniques. ACO has been extensively studied and deployed for solving problems as varied as Vehicular Routing Problem (VRP) (Rizzoli et al., 2004), Single Machine Total Weighted Tardiness Problem (SMTWTP) (Den Besten et al., 2000), Graph Colouring Problem (Vesel and Zerovnik, 2000), and so on. These facts have been attracting the attention of researchers studying distributed scheduling.

The rest of the work is organized as follows. The next Section provides more details on the concept of Parameter Sweep Experiments. Section "Distributed Computing Infrastructures" describes the two distributed environments commonly used today, namely Grids and Clouds. Later, Section "Swarm Intelligence" explains the concepts underpinning Swarm Intelligence and Ant Colony Optimization. Section "Related work of job scheduling based on ACO" presents related works of job scheduling based on ACO. Section "Analysis of ACO-based Scheduling Approaches" present a uniform and organized view of the surveyed works. Finally, Section "Conclusions and Future Research Directions" concludes this work and describes prospective future works.

BACKGROUND

Parameter Sweep Experiments (PSEs) is an experimental simulation-based methodology involving running the same application code several times with different input parameters to derive different outputs (Youn and Kaiser, 2010). Running PSEs requires managing many independent jobs (Samples et al., 2005), since the experiments are executed under multiple initial configurations (input parameter values) a large number of times, to locate a particular point in the parameter space that satisfies certain user criteria. In addition, different PSEs have different number of parameters.

Scientists involved in this type of experiments need a computing environment that delivers large amounts of computational power over a long period of time. In general terms, such an environment is called a High Throughput Computing (HTC) environment. In HTC, jobs are dispatched to run independently on multiple computers at the same time. Interestingly, PSEs find their application in diverse scientific areas such as Bioinformatics (Sun et al., 2004), Earth Sciences (Gulamali et al., 2004), High-Energy Physics (Basney et al., 2000), Molecular Science (Wozniak et al., 2005) and even Social Sciences (Axelrod, 1997). However, to deal with these problems, it is necessary large amounts of computational power.

A concrete example of PSE is the one presented by Careglio et al. (Careglio et al., 2010), which consists in analyzing the influence of size and type of geometric imperfections in the response of a simple tensile test on steel bars subject to large deformations. To conduct the study, the authors numerically simulate the test by varying some parameters of interest, namely using different sizes and types of geometric imperfections. By varying these parameters different study cases were obtained, which was necessary to analyze and run on different machines in parallel.

Due to the fact that PSEs involve three broad activities, namely the execution of a potentially large computation, the grouping of the results, and their interpretation afterwards, PSEs can be carried out by exploiting computational workflows (Taylor et al., 2006). Therefore, PSEs, which are embarrassingly parallel problems, are well suited to HTC environments where large numbers of resources are available. To this end, PSEs require a suitable partition of the input data where each partition is assigned to a different job (or sub

workflow) (Berglund et al., 2009; Ma and Buyya, 2005). Even though PSEs can be done without workflows, data management in the client is simplified when the parameter sweep study can be treated as several sub workflows. This increased abstraction is beneficial both from a usability point of view (i.e. results interpretation becomes faster) and of course from a performance standpoint.

When designing PSEs, there are several issues to tackle. On one hand, it is necessary to generate the set of all possible combinations of input parameters. This is a time-consuming task, which should be automated. However, it is not straightforward to provide a general solution, since each problem has a different number of parameters and each of them has its own variation interval. Another issue, which is in part a consequence of the first issue, relates to scheduling PSEs on distributed environments, which is a complex activity. For this reason, it is necessary to develop efficient scheduling strategies to appropriately allocate the workload and reduce the associated computation time. The term scheduling refers to the way jobs are assigned to run on the available CPUs, since there are typically many more jobs running than available CPUs. This assignment is carried out by software known as a scheduler and dispatcher. The demand for scheduling is to achieve high performance computing. The scheduling problem is defined NP-complete problem (Woeginger, 2003) and it is not trivial.

In PSEs scheduling problems, in order to minimize the completion time of jobs (makespan) it is essential a correct assignment of jobs so that computer loads and communication overheads are well balanced (load balancing). The term "makespan" means to find a sequence of jobs that minimizes the finishing time of the last job in the system, or in other words the maximum completion time of all jobs. A formal definition of makespan is as follows: given n jobs with processing times $\{p1, p2, ... pn\}$ and m machines with speeds $\{s1, s2, ..., sm\}$ we want to assign the jobs to machines to minimize the maximum finish time. On the other hand, load balancing (Yawei and Zhiling, 2004) refers to the technique that tries to distribute work load between several computer resources (CPUs, network interfaces, hard drives, and other resources) in order to get an optimal resource utilization, throughput, or response. A load balancing mechanism aims to equally distribute the load on each computing node, maximizing their utilization and minimizing the total job execution time. To achieve these goals, any load balancing mechanism should be fair in distributing the load across the computing nodes. Note that these two concepts are mutually independent, and in fact represent a trade-off, since a good load balancing does not always lead to a minimization of completion time of all jobs and viceversa.

Distributed Computing Infrastructures

Years ago, Grid Computing (Foster and Kesselman, 2003) and more recently Cloud Computing technologies (Buyya et al., 2009) have been increasingly used for running parameter sweep applications. PSEs are well suited for these environments since PSEs are inherently parallel problems with no or little data transfer between nodes during computations. Since many applications require a great need for calculation, these applications have been initially targeted at dedicated High Throughput Computing (HTC) infrastructures such as clusters or pools of networked machines, managed by some software or platform such as Condor (Thain et al., 2005).

Then, with the advent of Grid Computing new opportunities were available to scientists, since Grids offered the computational power required to perform even larger experiments. Grid Computing introduced new facilities such as dynamic service discovery, the ability of relying on a large number of resources belonging to different administrative domains, and finding the best set of machines that meet an application's requirements.

The use of Grid Computing in scientific applications (Coveney et al., 2005) has been successful in many international projects and has led to the establishment of world-wide infrastructures available for computational science (Pordes et al., 2007; Gagliardi and Begin, 2005; Catlett, 2005).

Cloud Computing is a natural evolution of the widespread adoption of virtualization, service-oriented architectures, and utility computing (Buyya et al., 2009). Basically, technological details are abstracted from end-users, who no longer have need for expertise in, or control over, the technology infrastructure "in the cloud" that supports them. Cloud Computing describes a new supplement, consumption, and delivery model for IT services based on Internet protocols, which typically involves provisioning of dynamically scalable and often virtualized resources.

Due to the fact that Grid Computing and Cloud Computing are nowadays the most used Infrastructures to execute scientific applications, an overview of each one is provided next.

Grid Computing

The term "Grid Computing" originated in the early 1990s as a metaphor of making computer power as easy to access an electric power Grid (Foster and Kesselman, 2003). Grid Computing can be defined as a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous and heterogeneous resources dynamically at runtime depending on their availability, capability, performance, cost, and user's Quality-of-Service (QoS) requirements (Baker et al., 2002).

A Grid, or the kind of distributed infrastructure that is built by following the Grid Computing paradigm, is a form of distributed computing whereby a "super virtual computer" is composed of many networked, loosely coupled computers acting together to execute very large jobs. As such, a Grid is a shared environment implemented via the deployment of a persistent, standards-based service infrastructure that supports the creation of, and resource sharing within, distributed communities. Resources can be computers, storage space, instruments, software applications, network interfaces and data, all connected to a network (private, public or the Internet) through a middleware software layer that provides basic services for security, monitoring, resource management, and so forth. Resources owned by various administrative organizations are shared under locally defined policies that specify what is shared, who is allowed to access what, and under what conditions (Foster and Iamnitchi, 2003). Basically, the problem that underlies the Grid concept is achieving coordinated resource sharing and problem solving in dynamic, multi-institutional Virtual Organizations (VO) (Foster et al., 2001) where each VO can consist of either physically distributed institutions or logically related projects/groups. The goal of such an infrastructure is to enable federated resource sharing in dynamic, distributed environments.

Grid Computing has been applied to computationally intensive scientific, mathematical, and academic problems, and it is used in commercial enterprises for such diverse applications as drug discovery, economic forecasting, seismic analysis, and back office data processing in support for e-commerce and Web services. Grids provide a means for offering information technology as a utility for commercial and non-commercial clients, with those clients paying only for what they use, as with electricity or water.

Despite the widespread use of Grid technologies in scientific computing, as demonstrated by the large amount of projects served by Grid Computing (Vecchiola et al., 2009), some issues still make the access to this technology not easy for disciplinary or domain users. For example, operationally, some Grids are bureaucratic, since research groups have to submit a proposal describing the type of research they want to carry out prior to executing their experiments.

Other usage-related issues involve technical hurdles. In most cases scientific Grids feature a prepackaged environment in which applications will be executed. Then, specific tools and APIs have to be used, and there could be limitations on the hosting operating systems or on the services offered by the runtime environment. On the other hand, although Grid Computing favors dynamic resource discovery and provision of a wide variety of runtime environments for applications, in practice, a limited set of options are available for scientists, which are not in addition elastic enough to cover their needs. An illustrative example involves the use of specific software that could not be available in the runtime environment were applications are executed. In general, applications that run on scientific Grids are implemented as bag of jobs applications, workflows, and MPI (Message Passing Interface) (Gropp et al., 1994) parallel jobs. Some scientific experiments could not fit into these models and therefore have to be reorganized or redesigned to exploit a particular scientific Grid.

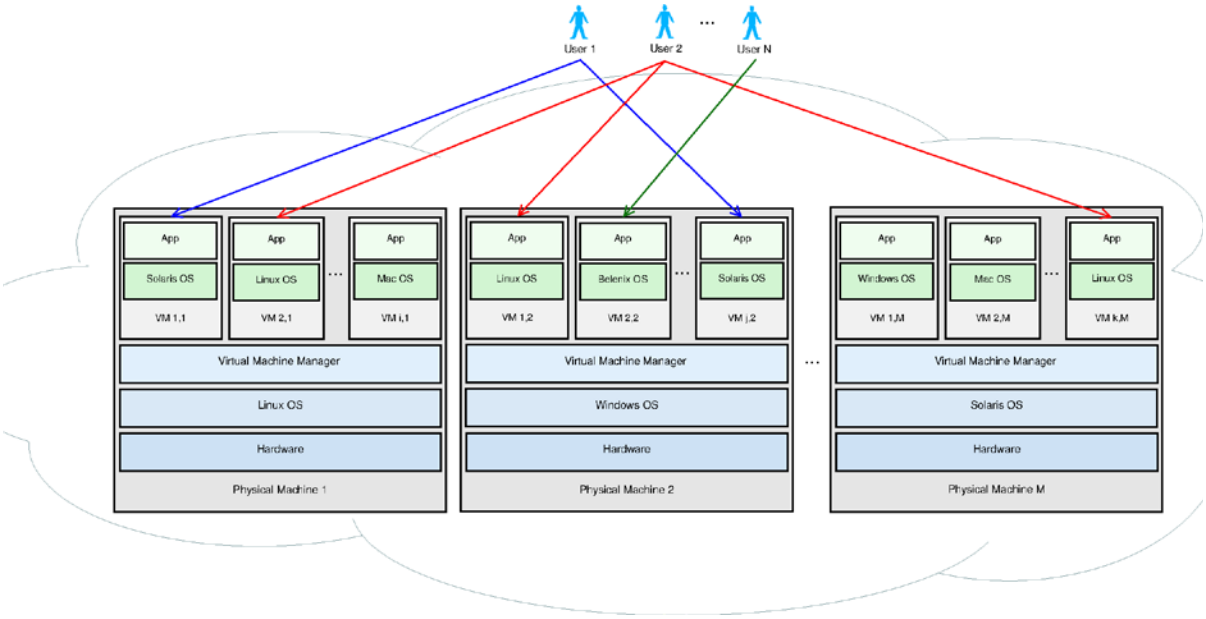
Cloud Computing

All in all, while the aforementioned bureaucratic issues can be a minor problem, the technical ones could constitute a fundamental obstacle for next generation scientific computing. Cloud Computing (Buyya et al., 2009), the current emerging trend in delivering IT services, has been recently proposed to address the aforementioned problems. By means of virtualization technologies, Cloud Computing offers to end users a variety of services covering the entire computing stack, from the hardware to the application level, by charging them on a pay per use basis. This makes the spectrum of options available to scientists, and particularly PSEs users, wide enough to cover any specific need from their research. Another important feature, from which scientists can benefit, is the ability to scale up and down the computing infrastructure according to the application requirements and the budget of users. By using Cloud-based technologies scientists can have easy access to large distributed infrastructures and are allowed to completely customize their execution environment, thus deploying the most appropriate setup for their experiments. Moreover, by renting the infrastructure on a pay per use basis, they can have immediate access to required resources without any capacity planning and they are free to release resources when these latter are no longer needed.

As suggested, central to Cloud Computing is the concept of virtualization, i.e. the capability of a software system of emulating various operating systems. By means of this support, scientists can exploit Clouds by requesting from them machine images, or virtual machines that emulate any operating system on top of several physical machines, which in turn run host operating systems. Usually, Clouds are established using the machines of a datacenter for executing user applications while they are idle.

Interaction with a Cloud environment is performed via Cloud services (Buyya et al., 2009), which define the functional capabilities of a Cloud, i.e. machine image management, access to software/data, security, and so forth. Cloud services are commonly exposed to the outer world via Web Services (Hao et al., 2010), i.e. reusable software components that can be remotely invoked by applications implemented in any programming language. By using these services, a user (scientific) application can allocate machine images, upload input data, execute, and download output (result) data for further analysis. Finally, to offer on demand, shared access to their underlying physical resources, Clouds have the ability to dynamically allocate and deallocate machines images. Besides, and also important, Clouds can coallocate N machines images on M physical machines, with $N \geq M$, thus concurrent user-wide resource sharing is ensured. These relationships are depicted in Figure 1.

Figure 1. Cloud computing: High-level view



In summary, a Cloud gives users the illusion of a single, powerful supercomputer in which complex applications can be run. Besides, the software stack of the infrastructure can be fully adapted and configured according to users' needs. This provides excellent opportunities for scientists and engineers to run applications that demand by nature a huge amount of computational resources -i.e. CPU cycles, memory and storage- and rely on specific software libraries.

With everything mentioned so far, there is a great consensus on the fact that from the perspective of domain scientists the complexity of traditional distributed and parallel computing environments such as clusters and particularly Grids should be hidden, so that domain scientists can focus on their main concern, which is performing their experiments. As a result, the use of Cloud Computing infrastructures is a good choice for running scientific applications. Precisely, for parametric studies, or scientific applications in general, the value of Cloud Computing as a tool to execute complex applications has been already recognized within the scientific community (Wang et al., 2008).

While Cloud Computing helps scientific users to run complex applications, job management is a key concern in Cloud Computing that must be addressed. Broadly, job scheduling is a mechanism that maps jobs to appropriate resources to execute, and the delivered efficiency will directly affect the performance of the whole Cloud Computing environment. Particularly, the scheduling algorithms for distributed systems have the goal of dividing a single computation into several jobs and submitting these latter to resources, while maximizing resource utilization and minimizing the total execution time (makespan) of all jobs. As job scheduling is an NP-complete optimization problem, several heuristic algorithms have been proposed in the literature.

Moreover, as artificial life techniques have shown to be useful in optimization problems, they are good candidates to optimize in principle load balancing and minimize the total execution time (makespan) of jobs in PSE environments. The advantage of these techniques derives from their ability to explore solutions in large search spaces in a very efficient way.

Swarm intelligence

As suggested earlier, Swarm Intelligence (SI) techniques are increasingly used to solve optimization problems. The expression Swarm Intelligence was introduced by Gerardo Beni and Jing Wang in 1989 in the context of cellular robotic systems (Beni and Wang, 1989) and concerns a type of problem solving skill inspired by nature. Swarm Intelligence (Bonabeau et al., 1999) is the discipline that deals with natural and artificial systems composed of many individuals that coordinate themselves using decentralized control and self-organization. In particular, the discipline focuses on the collective behaviors that result from the local interactions of the individuals with each other and with their environment. Examples of systems studied by SI are ants colonies, fish schools, flocks of birds, and herds of land animals, where the whole group of agents perform a desired chore (i.e. feeding), which may not be made individually. Figure 2 shows some examples of natural systems which are inspired by Swarm Intelligence. For example, an individual ant is relatively unintelligent, but when they are part of a colony, complex group behavior emerges from the interactions of individuals -such as searching for food- who exhibit simple behaviors by themselves. On the other hand, a fish makes dynamic decisions to swim in one direction or another. At beginning, a fish swims behind another performing various maneuvers, but only up to a certain point. If the first fish swims right passing food, the other members of the school would listen to other instincts instead.

Swarm intelligence is then the emergent collective intelligence of groups of simple autonomous agents, where an autonomous agent is a subsystem that interacts with its environment, which probably consists of other agents, but acts relatively independently from all other agents. The autonomous agent does not follow commands from a leader, or some global plan. For example, for a bird to participate in a flock, it only adjusts its movements to coordinate with the movements of its flock mates, typically its neighbors that are close to the bird in the flock. A bird in a flock simply tries to stay close to its neighbors, but avoid collisions with them. No bird takes the leadership role since there is no lead bird.

Any bird can fly in the front, center and back of the "swarm". Swarm behavior helps birds taking advantage of several things including protection from predators (especially for birds in the middle of the flock), and searching for food (essentially each bird is exploiting the eyes of every other bird). The essential principle in SI is cooperation and exchange of knowledge between individual information units.

An SI system has the following properties:

- It is composed of many individuals;
- The individuals are relatively homogeneous, i.e. they are either all identical or they belong to a few typologies;

Figure 2. Examples of natural systems

- The interactions among the individuals are based on simple behavioral rules that exploit only local information that the individuals exchange directly or via the environment;
- The overall behavior of the system results from the interactions of individuals with each other and with their environment, i.e. the group behavior self-organizes.

The characterizing property of a SI system is its ability to act in a coordinated way without the presence of a coordinator or of an external controller. Most often, the behavior of each individual of the swarm is described in probabilistic terms, as each individual has a stochastic behavior that depends on his local perception of the neighborhood.

The following Section explains Ant Colony Optimization (ACO), an SI technique that is widely used in job scheduling problems, and will be the basis for analyzing the state of the art of the subject in this work.

Ant Colony Optimization

Ant Colony Optimization algorithm (ACO), introduced by Marco Dorigo in 1992 in his doctoral thesis (Dorigo, 1992), is a probabilistic technique for solving computational problems which can be reduced to finding shortest paths through graphs. ACO was inspired by the observation of real ant colonies. An interesting behavior is how ants can find the shortest paths between food sources and their nest.

In the real world, ants (initially) wander randomly, and upon finding food return to their colony while laying down pheromone trails. If other ants find such a path, they are likely not to keep traveling at random, but to instead follow the trail, returning and reinforcing it if they eventually find food. Thus, when one ant finds a good (i.e. short) path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leaves all the ants following a single path. The idea of ACO is to mimic this behavior with *simulated ants* walking around the graph representing the problem to solve.

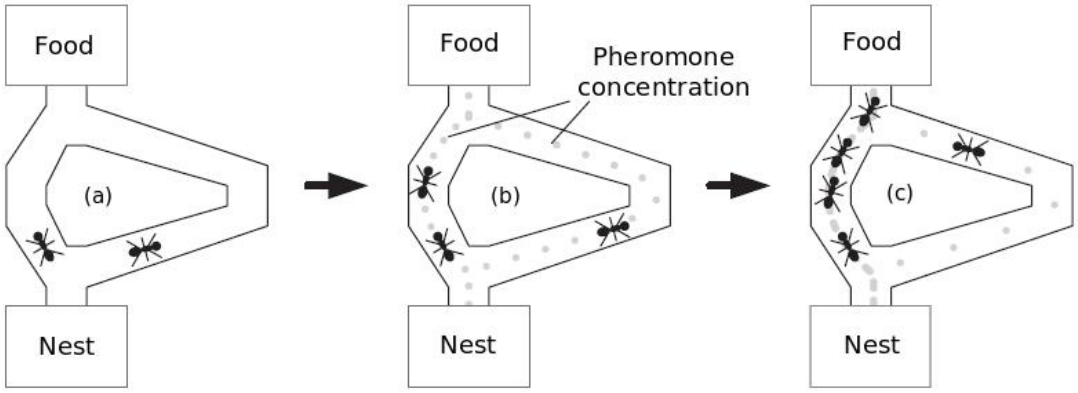
Over time, however, pheromone trails start to evaporate, thus reducing their attractive strength. The more the time it takes for an ant to travel down the path and back again, the less the frequency with which pheromone trails are reinforced. A short path, by comparison, gets marched over faster, and thus the pheromone density remains high as it is laid on the path as fast as it can evaporate.

From the algorithmic point of view, the pheromone evaporation process has also the advantage of avoiding the convergence to a locally optimal solution. If there were no evaporation at all, the paths chosen by the first ants would tend to be excessively attractive to the following ones. In that case, the exploration of the solution space would be constrained. Thus, when one ant finds a good path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leaves all the ants following a single path.

Figure 3 shows two possible paths from the nest to the food source, but one of them is longer than the other one. Ants will start moving randomly to explore the ground and choose one of two ways as can be seen in (a). The ants taking the shorter path will reach the food source before the others and leave behind them a trail of pheromones. After reaching the food, the ants will turn back and try to find the nest. The ants that go and return faster will strengthen more quickly the pheromone amount in the shorter path, as shown in (b). The ants who took the long way will have more probability to come back using the shortest way, and after some time, they will converge toward using it. Consequently, the ants will find the shortest path by themselves, without having a global view of the ground. After a certain time, almost all ants will choose the left path as shown in (c).

Precisely, the above behavior of real ants has inspired ACO, which is a population-based approach that has been in turn successfully applied

Figure 3. Adaptive behavior of ants (adapted from Ciruela Martín, 2008)



to many NP-hard optimization problems (Dorigo and Caro, 1999). One of its main ideas is exploiting the indirect communication among the individuals of an ant colony. Intuitively, this mechanism is based on an analogy with the abovementioned trails of pheromone which real ants use for communication. ACO employs pheromone trails as a kind of distributed numerical information which is modified by ants to reflect their accumulated experience while solving a particular problem.

When applied to optimization problems, ACO uses a colony of artificial ants that behave as cooperative agents in a solution space were they are allowed to search and reinforce pathways (solutions) in order to find the optimal ones. A solution that satisfies the problem constraint is feasible. After initialization of pheromone trails, ants construct feasible solutions, starting from random nodes, and then pheromone trails are updated. A node is an abstraction for the location of an ant, i.e. a nest or a food source. At each execution step ants compute a set of feasible moves and select the best one (according to some probabilistic rules) to carry out the rest of the tour. The transition probability is based on the heuristic information and pheromone trail level of the move. The higher the value of the pheromone and the heuristic information, the more profitable it is to select this move and resume the search. In the beginning, the initial pheromone level is set to a small positive constant value *0* and then ants update this value after completing the construction stage. All ACO algorithms adapt a specific algorithm scheme as shown in Figure 4.

After initializing the pheromone trails and control parameters, a main loop is repeated until the stopping criterion is reached. The stopping criterion can be for example a certain number of iterations or a given time limit without improving the overall result. In the main loop the ants con-

Figure 4. Pseudo-code of a basic ACO algorithm

```
Procedure ACO
Begin
  Initialize the pheromone
  While (stopping criterion not satisfied) do
    Position each ant in a starting node
    Repeat
      For each ant do
        Chose next node by applying the state transition
      End for
    Until every ant has built a solution
    Update the pheromone
  End while
End
```

struct feasible solutions and update the associated pheromone trails. More precisely, partial problem solutions are seen as nodes: each ant starts from a random node and moves from a node *i* to another node *j* of the partial solution. At each step, the ant *k* computes a set of feasible solutions to its current node and moves to one of these expansions, according to a probability distribution. For an ant *k* the probability pkij to move from a node *i* to a node *j* depends on the combination of two values:

$$\begin{cases} p_{ij}^{k} = \dfrac{\tau_{ij}\eta_{ij}}{\sum q.allowed_k \tau_{iq}\eta_{iq}} & if \ j \in allowed_k \end{cases}$$

where:

- η_{ij} is the attractiveness of the move as computed by some heuristic information indicating a prior desirability of that move;
- τ_{ij} is the pheromone trail level of the move, indicating how profitable it has been in the past to make that particular move (it represents therefore a posterior indication of the desirability of that move);
- $allowed_k$ is the set of remaining feasible nodes.

Thus, the higher the pheromone value and the heuristic information, the more profitable it is to include state j in the partial solution. The initial pheromone level is set to *τ0*, which is a small positive constant. In nature there is not any pheromone on the ground at the beginning, or the initial pheromone is *τ0 = 0*. If in the ACO algorithm the initial pheromone is zero, then the probability to chose next state will be *pkij = 0* and the search process will stop from the beginning. For this reason, the initial pheromone must be a positive numeric value.

Furthermore, the pheromone level of the elements of the solutions is changed by applying the following updating rule:

$$\tau_{ij} \leftarrow \rho.\tau_{ij} + \Delta\tau_{ij}$$

where the inequality $0 < \rho < 1$ models evaporation and $\Delta\tau_{ij}$ is an additional pheromone and it is different for each implementation of ACO algorithms. Normally, the quantity of the added pheromone depends on the quality of the solution.

In practice, to optimize job scheduling problems, the ACO algorithm has the advantage that allows the use of graphical representations, where a graph design is used to identify the problem and connect the corresponding arcs of each job to each submitting physical machine. Here, each job can be represented by an ant, i.e. by an agent. Agents cooperatively search the less-loaded machines with sufficient available resources and transfer the jobs to these machines.

RELATED WORK OF JOB SCHEDULING BASED ON ACO

In the last ten years, Swarm Intelligence has received increasing attention in the research community. Due their fast convergence rate, global optimization ability and robustness, SI algorithms were applied to approximate classical NP-complete problems such as Traveling Salesman Problem (TSP) (Bianchi et al., 2002; Kuo et al., 2010), Job Shop Problem (JSP) (Heinonen and Pettersson, 2007; Huang and Liao, 2008) and other optimization problems (Yun-Chia and Smith, 2004; Baterina and Oppus, 2010).

Within the realm of combinatorial optimization, SI finds its niche in routing applications and in specialized job scheduling activities. Not surprisingly, these two applications correlate very well with two fundamental traits of SI, i.e. positive

feedback or reinforcing good solutions present in the system, and division of labor. Moreover, social insect colonies can solve complex problems collectively by distributed and intelligent methods. These problems are beyond the capabilities of each individual insect, and the cooperation among them is largely self-organized. As a result, the collective behavior of insects has become a model for tackling job scheduling problems.

Particularly, in recent years, several researchers have proposed algorithms based on ACO for solving job scheduling problems in distributed environments, particularly Grids and Clouds. After analyzing the existing literature, we have classified the relevant works into four main groups according to the objectives (or variables) the associated scheduling algorithms are designed to optimize:

- Approaches minimizing makespan
- Approaches maximizing load balancing
- Approaches minimizing makespan and maximizing load balancing
- Approaches minimizing makespan and minimizing monetary cost
- Approaches minimizing makespan, maximizing load balancing and minimizing monetary cost

In next subsections, the approaches for job scheduling based on ACO falling in these categories are discussed. Within each category, approaches are referenced by using authors' names. Lastly, the fourth and fifth categories represent scheduling algorithms that assume distributed environments which charge users for the computational resources (typically CPU time and network usage) they spend when running their applications.

Approaches Minimizing Makespan

Lorpunmanee et al.

In (Lorpunmanee et al., 2007) the authors have proposed an ACO algorithm for dynamic job scheduling in Grid environments where the availability of resources is constantly changing and jobs arrive to be executed at different times. Generally, jobs are sent to a Grid at different time points, and each job has different lengths and consumes different resources. Therefore, job processing is performed at different execution times.

The proposed ACO algorithm takes into account the requirements of each job, which are independent of each other. Moreover, each processor execute only one job per unit time, is to say that once a processor finishes to execute a job can process a new job. The motivation of this paper was to develop an ACO algorithm that can produce an optimal selection of resources and minimize efficiently and effectively the total tardiness time (makespan), i.e. to improve the overall performance of a set of jobs within a dynamic Grid.

In the algorithm, the authors have defined the completion time (*CT*) as the wall clock time time, which machine completes for each job. The completion time of job *jth* in a machine *i* is defined as $C_{ij} = aj + rj + ETC_{ij}$, where a_j is the arrival time of the job *j*, *rj* is the release time of the job *j*, and *ETCij* is defined as the amount of time that the job *j* is processed in the machine *i*, and the machine has no load to the assigned job. Moreover, the algorithm includes four steps, namely pheromone initialization, state transition rule, local update rule and global update rule.

Initially, a set of artificial ants are created. Each ant starts with one unscheduled job in one machine and then the ant builds a job tour in machines until a feasible solution is constructed. To do this, the ant makes the best possible move as marked by pheromone trails and the heuristic information. To do this ants use the state transition rule. The heuristic is used to determine the desirability of to move the job *j* from machine *i* to machine *m*. This is inversely proportional to the completion time of the job *j* that has been assigned on machine *m*, or $\eta j(i, m) = 1=(aj + rj + Pm,j)$, where *P* is the processing time of the job *j* on the machine *m*. The local update rule is used

by ants while building a solution. Ants visit paths and change their pheromone level which is used immediately to locally update the rule. The local rule reduces the convergence because ants choose a new machine based on high pheromone levels.

A machine becomes less desirable for the following ants, if the pheromone trail is reduced. The global update rule is performed after that each ant has completed its tour (a feasible solution) and only one ant that has the best solution found so far, is allowed to deposit pheromone to the path after each iteration. Therefore, job *j* is assigned from machine *i* to machine *m* in the global best solution.

The experiments performed to evaluate the algorithm were simulated using the GridSim (Buyya and Murshed, 2002) toolkit. The results obtained by authors have shown that the Ant Colony Optimization algorithm is the best average case of the tardiness time with respect to other algorithms as First Come First Served (FCFS), Minimum Time Earliest Due Date (MTEDD) and Minimum Time Earliest Release Date (MTERD). FCFS is a policy whereby the requests of service are attended to in the order that they arrived, without other biases or preferences. MTEDD orders the sequence of jobs to be serviced from the job with the earliest due date to the job with the latest due date. Finally, MTERD gives the highest priority to the job that has the earliest release date in the queue.

Mathiyalagan et al.

Another improved ACO algorithm was proposed in (Mathiyalagan et al., 2010b). Authors have developed a modified pheromone updating rule which solves the Grid scheduling problem more effectively than the traditional ACO algorithm (Dorigo and Caro, 1999) by modifying its basic pheromone updating rule. The basic pheromone updating rule $\tau_{ij}(t)new= \rho..\tau_{ij}(t)old + \Delta\tau_{ij}(t)$ has been changed $\tau_{ij}(t)new=(\rho.\tau_{ij}(t)old)+(\rho/(\rho+1).\Delta\tau_{ij}(t))$, where τij is the trail intensity of the path *(i, j)* (*j* is a job and *i* is the machine assigned to the job *j*), ρ is an evaporation rate and $\Delta\tau ij$ is an additional pheromone added when a job is moved by the scheduler to a resource.

The experimental results carried out by the authors prove that the improved ACO algorithm has an effective role on Grid scheduling. The modified pheromone updating rule makes the improved ACO algorithm to work more efficiently than the original ACO algorithm. This approach was also simulated using GridSim (Buyya and Murshed, 2002) toolkit. The authors have achieved a better optimization level because the improved ACO algorithm finds the best resources to assign each job at a faster rate, thus increasing efficiency.

Benerjee et al.

Another heuristic approach proposed by Benerjee et al. (Banerjee et al., 2009) based on ACO was adapted to address service allocation and scheduling within a Cloud Computing environment. The proposed optimization method is mainly aimed to maximize the scheduling throughput to handle all the diversified requests according to different resource allocator available under a Cloud Computing environment. Second, the pheromone update mechanism has been modified to minimize the makespan of services based on Cloud Computing. Steps of the classic ACO algorithm have been modified to manage a Cloud architecture. A Cloud is viewed as a collection of clustered services for executing jobs and storing data, hence a live service of a Cloud behaves like an ant. Each path between machines *(r, s)* has a distance or cost associate $\delta(r, s)$ and a pheromone concentration $\tau(r, s)$. The pheromone updating rule is applied as:

$$\tau (r, s) = (1 - \alpha) \tau (r, s) + \Sigma \Delta\tau k (r, s)$$

where α is the pheromone evaporation factor between *0* and *1*, and $\Delta\tau k (r, s)$ is the cost done by ant *k* if *(r, s)* is its path and it is *0* if it is not in the path.

Every time a request is processed on a Cloud machine, the pheromone concentration is updated for all the paths between machines modifying the above formula by associating a parameter •τ.

$$\tau (r, s) = (1 - \alpha tcs)\ \tau(r, s) + \Sigma\Delta\ \tau k+tcs(r, s)$$

where •τ represents the time for each cloud scheduling service and αtcs accounts for the evaporation factor under time slot within cloud machines. The heuristic is divided into two categories for Cloud-based services: online mode service and batch mode service. In online mode, whenever a request arrives is immediately allocated to the first free resource allocator. The arrival order of the request in a Cloud is important in the proposed method. Each service request is considered only once for matching and scheduling. In batch mode, requests are collected and the scheduler considers the approximate execution time for each job. Then, the scheduler uses a heuristic to make a better decision. The authors have run examples in real Cloud environments by using Google App Engine and Microsoft Live Mesh in order to evaluate the proposed ACO algorithm, achieving performance improvements.

Ritchie and Levine

The work proposed in (Ritchie and Levine, 2004) describe an ACO algorithm that, when combined with Local Search (LS) (Alabas Uslu and Dengiz, 2011) and Tabu Search (TS) (Liao and Huang, 2011), can find shorter schedules on benchmark problems than other techniques described in (Braun et al., 2001). With LS technique, any solution s will have at least one processor with a schedule length equal to the makespan of the solution, which is called the "problem" processor. When more than one problem processor exists, one of them is picked arbitrarily. The neighborhood *Ne* of the solution *s* is defined as all solutions that differ by a single transfer of a job currently allocated from the problem processor to any other processor, or by a single swap of a job currently allocated to the problem processor with a job allocated to other processor. On the other hand, TS is a more sophisticated LS strategy that tries to avoid entrapment in local minimum by using a tabu list of previously visited regions of the search space and disallowing moves that would result in a solution that is contained in the list, i.e. one that has been seen before.

The goal of this work was to minimize the total execution time of a metatask (collection of independent jobs with no inter-job dependences, i.e. a PSE). Here, the authors assume that the expected running time of each individual job on each processor must be known. This information is stored in an "Expected Time to Compute" (*ETC*) matrix where a row contains the *ETC* for a single job on each one of the available processors. Moreover, any *ETC* matrix will have *n* x *m* entries, where *n* is the number of jobs and *m* is the number of processors.

In order to simulate various possible heterogeneous scheduling problems as realistically as possible, the authors have defined different types of *ETC* matrices as proposed in (Braun et al., 2001) according to three metrics: job heterogeneity, machine heterogeneity and *ETC* consistency. Job heterogeneity is the amount of variance possible among the execution times of jobs and has two possible values: *high and low*. Machine heterogeneity represents a possible variation of the running time of a particular job across all processors, and again has two values: *high and low*. In order to try to capture some other possible features of real scheduling problems, three different *ETC* consistencies were used: *consistent, inconsistent and semi-consistent*. An *ETC* matrix is said to be consistent if whenever a processor *pj* executes a job *ji* faster than another processor *pk*, then *pj* will execute all other jobs faster than *pk*. This behavior can be seen as modeling an heterogeneous system such as Grid Computing in which the processors differ only in their processing speed. An *ECT* matrix is inconsistent when a processor *pj* can

execute some jobs faster than *pk* and some other slower. Such an inconsistent *ETC* matrix could therefore simulate a real network layer in which there are different types of available machines. Finally, a semi-consistent *ETC* matrix is an inconsistent matrix which has a consistent sub-matrix of a predefined size, and therefore simulates for example a Grid that incorporates a sub-network of similar machines (but with different processor speeds), but also includes an array of different computational devices.

In this work the authors have determined what information they must encode in the pheromone trail to allow ants to share useful information about good solutions. Due to the fact that jobs run at different speeds on different processors, this information is useful to store information about good processors for each job. Therefore, the pheromone value *(i, j)* was selected to represent the favorability of scheduling a particular job *j* into a particular processor *i*. The ants build their own solution using both information encoded in the pheromone trail and also problem specific information in the form of an heuristic. Here, the heuristic used is Min-min (Etminani and Naghibzadeh, 2007), which suggests that the heuristic value of particular job *j* should be proportional to the minimum completion time of *j*. The minimum completion time *j* can be expected to finish on its best processor *pjbest*. To leave a pheromone trail the authors have used the Max-min Ant System (MMAS) described in (Stützle and Hoos, 2000). Basically, only the best ant *sbest* is allowed to leave pheromone. As a consequence, after each iteration, the search is much more aggressive and significantly improves the performance of ACO algorithms.

Due to the fact that other researchers (Dorigo and Stützle, 2003; Levine and Ducatelle, 2003) have demonstrated that ACO algorithms can often effectively be improved by combining them with local search (LS) techniques, in this work the authors have applied an ad-hoc LS technique to the ACO algorithm proposed. The LS procedure exhaustively analyses this neighborhood and selects the swap or transfer which reduces the maximum schedule length of the two processors involved the most. The process must be repeated until no further improvement is possible. On the other hand, when TS is used in conjunction with the ACO algorithm it is simply used for n iterations to try to improve the solution of the iteration best ant, which already have had LS applied to it.

Approaches Maximizing Load Balancing

Hui Yan et al.

The work in (Hui et al., 2005) focuses on an improved ACO algorithm for job scheduling in Grid Computing. The aim of this approach is focused on maximizing load balancing on machines. When a resource *j* enrolls into a Grid system, it is asked to submit its performance parameters, such as number of processors, processing capability of each processor, communication ability, etc. In the Grid, a resource finder tests these parameters for validation and initializes the trail intensity, which represents pheromones of real ants, for the resource *j* in the ACO algorithm. Here, $\tau i(0)$ is the trail intensity on path *i* from the scheduler to the corresponding resource *i* at time *0*. In the classic ACO the pheromone trail is changed by applying the following updating rule: $\tau i = \rho \tau i + \Delta \tau i$, where ρ models evaporation and $\Delta \tau i$ is an additional pheromone.

This algorithm adds a load balancing factor. The load balancing factor λi, which is related to the job finishing rate in the resource *i* in order to change the pheromone trail. This makes the job finishing rate at different resource being similar and the ability of the systematic load balancing is improved. The trail intensity has been changed to $\tau i = \rho \tau i + \Delta \tau i + C\lambda i$, where $C > 0$ is the coefficient of the load balancing factor. When more jobs are finished, the trail intensity increase, contrarily, when the jobs are not completed the trail intensity decreases. A simulation system was developed to test the ACO algorithm in a simulated Grid environment, which showed the feasibility of the approach.

Fidanova and Durchova

The work proposed in (Fidanova and Durchova, 2005) introduces a job scheduling algorithm for Grid Computing. The algorithm is focused on maximizing load balancing on machines. The aim of this approach was to develop a high throughput computing scheduling algorithm based on ACO, which means scheduling a set of independent jobs to increase the processing capacity of a system over a long period of time.

The proposed ACO algorithm incorporates the use of a function *free(i)* to report when a machine *i* is released. If a job *tj* is executed on a machine *mi*, then the beginning time of *tj* becomes $bj = free(i) + 1$ and the new value of the function *free(i)* becomes $free(i) = b_j + ET_{ij} = CT_{ij}$ after assigning the job *tj*, where ET_{ij} is the expected execution time of the job *tj* on the machine *mi* and *CTij* is the expected completion time. The heuristic information used by the algorithm is $\eta_{ij} = 1/free(i)$. The heuristic decides that if a machine is freed earlier, the corresponding node –in SI terms– will be more desirable. At the end of each iteration the objective function is calculated as $Fk = max(free(i))$ over the solution constructed by ant k. The additional pheromone trail added by an ant is $\Delta\tau_{ij} = (1 - \rho)/Fk$, where ρ models the evaporation factor. Hence, in the subsequent iterations the elements of the solution with less value of the objective function will be more desirable.

In the proposed algorithm two kinds of sets of job are needed: a set of scheduled jobs and a set of arrived and unscheduled jobs. When the set of scheduled jobs becomes empty the scheduled algorithm is started over the jobs from the set of unscheduled jobs. This guarantees that the machines will be fully loaded.

Zehua and Xuejie

In the work (Zehua and Xuejie, 2010) the authors have proposed a load balancing mechanism based on ACO and complex network theory in Open Cloud Computing Federation (OCCF), a federation that includes multiple Cloud providers devoted to create an uniform Cloud resource interface to users. Moreover, in the context of network theory, a complex network (Carpi et al., 2011) is a graph (network) with non-trivial topological features -features that do not occur in simple networks such as lattices or random graphs but often occur in real graphs. The study of complex networks is a young and active area of scientific research inspired largely by the empirical study of real-world networks such as computer networks and social networks. Some of recent studies were focused on the issue of whether the same principles could be applied to the development of the computer-network communication (Strogatz, 2001).

Load balancing is a very important goal to achieve in Cloud Computing due two reasons. First, Cloud providers must use load balancing in its own Cloud platform to provide a solution with high efficiency for the user. Second, a load balancing mechanism is needed to achieve low monetary costs and "infinite" resource pool for users.

In nature and in technology many systems consist of a large number of highly interconnected dynamical units. Despite the inherent differences, most of the real networks are characterized by the same topological properties, such as relatively small characteristic path lengths, high clustering coefficients, and so on. All these features make real networks radically different from regular lattices and random graphs, i.e. the standard models studied in mathematical graph theory. For this reason, the authors only have considered small-world property and scale-free distribution. A network is called a small-world network by analogy with the small-world phenomenon, which is popularly known as six degrees of separation. The small world hypothesis is the idea that two arbitrary people are connected by only six degrees of separation, i.e. the diameter of the corresponding graph of social connections is not much larger than six.

The first small-world network model (Strogatz, 2001), which through a single parameter smoothly interpolates between a random graph to a lattice. Their model demonstrated that with the addition of only a small number of long-range links, a regular graph, in which the diameter is proportional to the size of the network, can be transformed into a "small world" in which the average number of edges between any two vertices is very small (mathematically, it should grow as the logarithm of the size of the network), while the clustering coefficient stays large. It is known that a wide variety of abstract graphs exhibit the small-world property, e.g., random graphs and scale-free networks. Further, real world networks such as the World Wide Web and the metabolic network also exhibit this property.

On the other hand, a network is named scale-free when the probability that a machine selected uniformly at random has a certain number of links (degree), follows a particular mathematical function called a power law. The power law implies that the degree distribution of these networks has no characteristic scale. In contrast, network with a single well-defined scale are somewhat similar to a lattice in that every machine has (roughly) the same degree. In a network with a scale-free degree distribution, some vertices have a degree that is orders of magnitude larger than the average these vertices are often called "hubs", although this is a bit misleading as there is no inherent threshold above which a machine can be viewed as a hub. If there were such a threshold, the network would not be scale-free.

These two characteristics are considered by the authors to move ants, since the machine with a large degree may leads the ants (or the jobs) move more quickly towards the region where more resources may be found for the execution of the jobs.

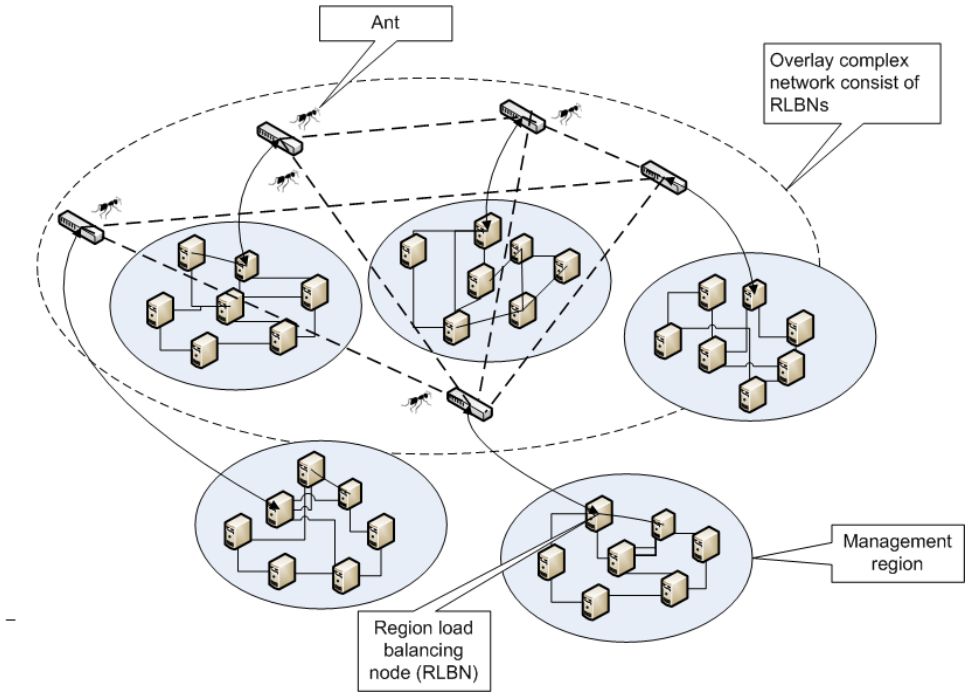
In order to perform the load balancing, the four following steps are carried out:

1. Underload load balancing step: Here an ant is periodically sent out by an underloaded machine to balance the workload on the whole OCCF and keep the complex network's vitality by updating the pheromone on each machine through the following steps: once the ant starts its trip from a machine and whenever the ant moves, during its trip, the ant remembers the machine which has maximum/minimum workload and the corresponding workload on them.
2. Overload load balancing step: Once a machine find its workload has excess its own threshold W, an ant is sent out by the machine, then, the following processes are carry out as the underload load balancing method except that the source machine of ants is appointed as the *Nmax*, where *Nmax* is the machine with maximum workload.
3. Pheromone update step: Once the load balancing is performed between the machines *Nmax* and *Nmin,* the ant backtracks the path that has traversed to update the pheromone values on the trail. To this end, each machine maintains a pheromone table of the trails which link to its neighbor machines and the values corresponding to the amount of pheromone on the path is stored in this table. The pheromone update process includes both increase and evaporation. Besides, the distribution of the pheromone changed on the whole path is assigned according to the strategies that more pheromone is assigned to the paths near to *Nmax* and less pheromone to the paths near to *Nmin*.
4. Complex network evolution step: The structure of the complex network evolves to adapt to changes in workload distribu-

tion, after updating the pheromone, adding a new path between machines *Nmax* and *Nmin* is considered only if there is not such a path and the cost effectiveness is greater than a threshold *Fmax*. On the contrary, the elimination of a path is considered if the path has been sleeping too long. Therefore, a complex network with the characteristic of small-world and scale-free is expected to be gained through the local behaviors of the colonies of the ants, theses two characteristic is useful in the load balancing processes of the ant algorithm. This means that the machine with a large degree in such a complex network may leads the ants move more quickly towards the region where more resources may be found for the execution of jobs.

Due to the fact a OCCF is consist of many cloud computing service provider's (CCSP) facilities, there would be many management regions -partitioned by geographically or the management strategies- that belong to an unique CCSP. A machine in each management region is chosen as the region load balancing node (RLBN), each RLBN connect with many of the other RLBNs of a CCSP according to the information get from the CCSP. After that, many RLBN of a CCSP are selected to connect with RLBNs in other CCSP, so the topology of the connected RLBNs makes an overlay network which can be regarded as a complex network. The structure of the OCCF is depicted in Figure 5. A RLBN can add to or remove from the management region. Also, another RLBN can be selected from the region machines once a RLBN is failed.

Figure 5. The structure of the OCCF and the formation of the complex network



Approaches Minimizing Makespan and Maximizing Load Balancing

Kousalya and Balasubramanie

In (Kousalya and Balasubramanie, 2009) the authors have proposed a modified ACO algorithm for Grid scheduling. The main focus of this work was to develop a high throughput scheduling algorithm based on ACO to minimize the makespan and maximize the resource utilization. The modified ACO is combined with Local Search (LS) and takes into consideration the available time of resources and the execution time of jobs to achieve a better resource utilization and a better scheduling. The LS (Alabas Uslu and Dengiz, 2011) technique is to define the neighborhood of a solution. In general a solution will have one or more resources, i.e. those with schedule lengths equal to the makespan of the whole solution.

In this work, before starting the grid scheduling, an expected execution time for each job on each machine must be estimated by an user and represented by an ET matrix. The ET matrix has *NxM* entries, where *N* is the number of independent jobs to be scheduled and *M* is the number of resources that are available. Each row of the *ET* matrix represents an estimated execution time for a job on each resource. Then, every column represents an estimated execution time for a particular resource. *ETij* is thus the expected execution time of the job *j* in the machine *i*.

The Ready time (*Readym*) indicates the time resource m would have finished the previously assigned jobs. The completion time *(CT)* of j^{th} job on the *i*th machine is $CT_{ij} = Readyi + ET_{ij}$. The proposed heuristic has two operating modes, an online mode where the scheduler is always ready, and other in batch mode, where the jobs and resources are collected and mapped at prescheduling time. In this sense, the bach mode scheduler takes better decisions because it knows details of available jobs and resources statically.

The number of jobs available for scheduling is always greater than the available number of machines in the grid. The free time of the machine *mi* is calculated using the function *free(i)*. The starting time of job *tj* on resource *mi* is *Bi = free(i) + 1* and then the new value of *free(i)* is the starting time plus ET_{ij}. A minimization function *F = max(free(i))* and the heuristic information $\eta i = 1/free(i)$ are used to find out the best resource. The pheromone level is updates adding an evaporation factor value between *0* and *1*, and an additional pheromone value.

The probability to move a job from a machine *i* to a machine *j* is computed by $P_{ij} = (\tau_{ij} . \eta_{ij} . (1/ CT_{ij})) / (\Sigma\tau_{ij} . \eta_{ij} . (1/ CTij))$. Here, the authors have included to the classic approach (Dorigo and Caro, 1999) CT_{ij} value, which represents the execution time of the *jth* job in the *ith* machine in the calculation of probability and has shown a positive result in performance improvement. This improvement is in terms of to decrease the makespan. The result produced by this algorithm is a little better than others algorithms proposed by the authors in (Kousalya and Balasubramanie, 2007; Kousalya and Balasubramanie, 2008) where *ETij* instead of CT_{ij} has been used.

The scheduling algorithm is executed periodically. Upon activation, the algorithm finds out the list of available resources (processors) in the Grid, forms the *ET* matrix and starts the scheduling. When all the scheduled jobs are dispatched to the corresponding resources, the scheduler starts to schedule over the unscheduled job matrix *ET*. This is to guarantee that the machines will be as loaded as possible.

An individual scheduling result in the modified ACO algorithm has four values (job, machine, starting time, completion time). These values are added to an output list. The starting time of the job *j* on the machine *i* is acquired through the function *free(i),* and the completion time is the starting time plus ET_{ij}. Finally, the output list is passed to an algorithm that uses the LS technique to reduce the overall makespan further. Try to

reduce the resource makespan will immediately reduce the overall makespan of the solution. The neighborhood is a solution of single transfer of a job from a resource to any other resources.

The experiments performed by the authors have shown that the proposed modified ACO algorithm is capable of producing high quality scheduling of jobs to Grid resources. Particularly, the algorithm can be used to design efficient dynamic schedulers for real time Grid environments. Additionally, by using ACO with their LS algorithm good load balancing results can be obtained.

Ruay-Shiung et al.

In (Ruay-Shiung et al., 2009) the authors propose the Balanced Ant Colony Optimization (BACO) algorithm for job scheduling in a Grid environment. The pheromone value on a path in the ant system is a weight for a resource in a Grid. Moreover, a resource with a larger weight value means that the resource has better computing power. For materializing the BACO algorithm the authors assume that each job is an ant and the algorithm sends the ants to search for resources.

BACO inherits the basic characteristics from ACO algorithm to decrease the computation time of executing jobs and considers the load of each resource. The BACO algorithm changes the pheromone density according to the status of resources by applying both a local pheromone update and a global pheromone update function. The pheromone value of each resource is stored in the scheduler. The pheromone indicator is calculated in each resource and for each job by adding an estimated transmission time and the execution time of a given job when it is assigned to a resource. The larger the value of pheromone *Pij* is, the more the efficiency of resource *i* when executing job *j*. The local and global pheromone update functions balance the system load. The local pheromone update function updates the status of the selected resource after a job assignment round. The global pheromone update function, on the other hand, updates the status of each resource for all jobs after the completion of each job.

Xu et al.

In the work proposed in (Xu et al., 2003), the authors validate the scalability of the ACO algorithm using a simple Grid simulation architecture for resource management and job scheduling. Once the authors have got the results of an n machines ACO problem, they can get the results of $n + m$ or $n - m$ machines of the same problem very quickly based on former results. The basic ACO algorithm has been improved by making it more suitable for Grid job scheduling. The extended algorithm works as follows:

1. When a resource *i* joins a Grid, the resource is asked to submit its performance parameters (number of Processing Elements, MIPS of every Processing Element, where MIPS (Zhang and Theodoropoulos, 2003) is a measure of a computer's central processing unit performance in Million Instructions Per Seconds, and so on). A resource monitor tests these parameters for validation and initializes links of pheromone taking into account these performance parameters.
2. Every time a new resource joins the Grid or a resource fails or a job is assigned or there is some job results available/returned, the pheromone value in the path to the corresponding resource will be changed. When a job is assigned to a resource, the transfer time of the job is subtracted to the pheromone value. When a job is returned successfully from a resource, an *encourage factor* is added to the pheromone value. When a job is returned as failed from a resource, a punish factor is applied.
3. The probability of assignment a job to a resource *i* is calculated taking into account the pheromone intensity on the path to the resource *i*, the innate performance of the resource (initial pheromone value), and two parameter values that correspond to the importance of the pheromone and the importance of the resource innate attributes.

In this work the authors have validated the scalability of the proposed algorithm by a simple Grid simulation architecture for resource management and job scheduling. To do several routing experiments, the authors have added new machines and providing or not providing to the algorithm previous information. When the extended algorithm uses previous information took less time to find the optimal or sub-optimal solutions. Using tens of machines or more than one hundred machine network, or cut off some machines of the network, the results shown similar findings. This characteristic is very helpful to Grid computing for its scalable and fault tolerance needs. The overall results have shown good response time and resource average utilization.

Ludwig and Moallem

In the work proposed in (Ludwig and Moallem, 2011) a distributed algorithm based on ACO (AntZ) is presented. Authors claim that to achieve a good load balancer the following characteristics should be addressed:

- **Optimum Resource Utilization:** The utilization of resources should be optimized by a load balancing algorithm to minimize time or cost related to these resources.
- **Fairness:** A load balancing algorithm should to be fair. This means that the difference between the heaviest loaded machine and lightest loaded machine in the network is minimized, keeping in mind that the search space is dynamic. The load is the number of jobs assigned to each resource relative to its computational power.
- **Flexibility:** When the topology of the network or the Grid changes, the algorithm should be flexible enough to adhere to these changes.
- **Robustness:** When failures in the system occur an algorithm should have a way to deal with the failure and be able to cope with the situation.

In other classical ACO-based approaches to job scheduling, ants act independently from jobs being submitted while in this approach there is a close binding between jobs and load balancing ants. In the AntZ algorithm, each job submitted to a Grid invokes an ant, which searches through the network to find the best machine to deliver the job. Ants leave information related to the machines that have visited as a pheromone trail. The pheromone trail in each machine helps other ants to find lighter resources more easily.

The AntZ algorithm adds a decay rate and a mutation rate to deal with the problem of load balancing.

When a job is submitted to a local machine in the Grid an ant is initialized and starts working. In each iteration, the ant collects the load information of the visited machine and save this information in its private history table. The ant also updates the load information table of the visited machines. The load information table of a machine contains information of its own load and load information of other machines, which were added to the table when ants visited the machine. 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 machines that ants have visited on their way to deliver their jobs together with their load information.

When an ant moves to the next machine has two choices. One choice is to move to a random machine with a probability given by the mutation rate. Another choice, on the other hand, is to use the load table information in the machine to choose where to go. The mutation rate decreases with a decay rate 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 –i.e. a predefined number of steps– is met. Finally, the ant delivers its job to the machine and dies.

The performance of the AntZ algorithm is evaluated using performance criteria such as makespan and load balancing level. In this re-

search authors have compared the performance of the AntZ algorithm with another SI algorithm based on Particle Swarm Optimization (ParticleZ) described later in this survey. The authors performed measurements to compare the two algorithms in order to identify which are more effective and under which conditions. The authors also compared the performance of the algorithms with other classical techniques. To carry out the experiments the GridSim [52] simulation toolkit was used.

The advantages of the proposed algorithms are threefold. First, the algorithms show good performance results and optimized resource utilization. Second, the algorithms have proved to be "fair" compared to a Random and State Broadcast Algorithm (SBA) approach. The Random approach is a simple scheduling algorithm in which the jobs being sent to the Grid are assigned randomly to different resources. On the other hand, SBA is based on broadcast messages which are exchanged between resources. Whenever the state of a machine changes, due to the arrival or departure of a job, the machine broadcasts a status message that describes its new state. This information policy enables each machine to hold its own updated copy of the system state Third, AntZ is very simple to implement which is a benefit for a distributed system. Finally, 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, which ensures scalability.

Palmieri and Castagna

The work presented by Palmieri and Castagna (Palmieri and Castagna, 2007) is focused on the ACO algorithm to achieve a good load balancing. In this work a Grid resource management framework was implemented as an ant-like self-organizing mechanism used to perform efficient resource management on Grid machines through a collection of very simple local interactions. These interactions are achieved by heuristically determining a scheduling solution that distributes the jobs on the Grid resources minimizing the overall Grid makespan, and the flowtime, i.e. sum of the completion times of all jobs. In this algorithm each job is carried by an ant. Ants cooperatively search for the less-loaded machines with sufficient available resources, and transfer the jobs to be executed to these machines. When an ant founds a machine to assign the job, it deposits pheromone to mark the detected solution.

A pheromone matrix will have a single entry for each job-machine pair in the problem and use a parameter which defines the pheromone evaporation rate. To build a solution the ants use heuristic information to guide their search with specific information of the problem. The heuristic value used by the ants for each job is inversely proportional to the minimum completion time for the job j on all the available machines, or better stated its completion time on the best available machine on the Grid.

Essentially, in this paper the authors want to maximize the productivity (throughput) of a Grid through an intelligent load balancing and at the same time, the authors want to obtain planning that offer a quality of service acceptable to the users. Consequently the fitness function for the assignment and balancing problem is simply be the inverse of the sum of makespan *ms* and mean flowtime *fs* of the solution *s*, weighted by a properly crafted parameter λ to give more priority to makespan, as it is the most important parameter. The fitness function equation is $fs = 1/(\lambda.ms + (1 - \lambda).fs)$.

Simulation results performed by authors upon different experimental Grid topologies have indicated that this proposed approach is highly adaptive, robust and effective in handling the above scheduling/load balancing problem. Moreover, the approach performs slightly better than the Tabu Search heuristic, especially in the presence of larger problems, i.e. with more Grid machines and jobs, since in these cases the number of ants associated to jobs and machines greatly increases.

Approaches Minimizing Makespan and Minimizing Monetary Cost

Srinivasan et al.

In (Srinivasan et al., 2005) the authors have proposed the Swarm Intelligence based approach for Task Allocation (SITA) algorithm. Here, the authors employed an ACO algorithm which aims is to optimize the often conflicting parameters of cost measured in Grid$ and execution time, both within a similar threshold. By programming a mathematical model of the behavior of the ACO into mobile agents (Garrigues et al., 2010), the authors proposed an heuristic for the Grid job allocation problem.

In the algorithm, a Global Resource Manager (GRM) accepts jobs from users. GRM maintains two queues. The first one is used for jobs with specification of time optimization, and the second one is devoted to jobs that require cost optimization. When a job is received by the GRM, it is queued into the appropriate queue depending on the scheduling policy specified by the user. In the Grid architecture implemented and used for give support this algorithm there is a layer below the GRM called Local Resource Managers (LRMs), which hold administrative authority over a subset of Grid resources that registered with it. Ants are deployed from a LRM to its corresponding β Grid, or a sub-grid under the administrative influence of the LRM. Each ant carries the job characteristics, the machines visited so far, the route cost so far and the Grid$ spent so far. An ant also maintains a tabu list which contains a list of visited paths. This list is maintained so that ants can avoid traveled links and thus avoid cycles. The characteristics include, trail intensity of both cost pheromone and time pheromone and the communication cost as obtained by using a modified link state flooding approach.

The next hop is selected probabilistically based on the pheromone intensity and the cost of that link. The pheromone evaporation rate makes the system responsive to dynamic network conditions, i.e. available network bandwidth, number of available machines, traffic network, etc. The job that was removed from the queue is handed over to all the LRMs which then individually compute the best allocation for this job making use of SITA. Based on an optimality index for the goodness of the fit, the best allocation is chosen for the job.

From the LRM and for each job, a swarm of explorer ants is deployed. The swarm crawls towards the best possible Grid resource to allocate the job to. Each ant chooses its next hop on the basis of a stochastic function that depends on two parameters, namely the proximity of a Grid Resource (to keep communication cost and transmission time low), and the trail intensity, which is a function of the number of ants that have gone through that link. In the LRM, after a specified percentage of ants report back the same path, the allocator ant is sent to the chosen Grid resource to allocate memory and resources. As the allocator ant traces the path, it proportionately lowers the pheromone levels.

Approaches Minimizing Makespan, Maximizing Load Balancing, and Minimizing Monetary Cost

Sathish and Reddy

In (Sathish and Reddy, 2008) the authors have presented and evaluated a dynamic scheduling strategy that maximizes the utilization of a Grid resource processing capabilities (load balancing), and reduces the processing cost and processing time taken to execute jobs on the Grid. This job scheduling strategy enhances (Xu et al., 2003) by taking into account the processing requirements for each job, the current state of the available resources, the current load and capacity of those resources, and the processing cost of those resources. The scheduler schedules a job based on the execution possibilities of the resources.

The possibility of assignment of a job to a resource *i* is calculated taken into account the pheromone intensity on the path to resource *i*, and two factors that represent the importance of the pheromone and the importance of the resource, respectively. The problem with the proposed algorithm in (Xu et al., 2003) is that it may schedule a job to a resource with low possibility even if the resources with high possibility are free, where "possibility" refers to a numeric value that tells how good a resource is for a given job. If the jobs are always scheduled to a resource with high possibility, then the load on the resource may be increased and the jobs may be kept waiting in the queue waiting for the resource to be free even though the other resources are free. To avoid this problem, Sathish and Reddy have proposed the following: if the difference between the possibility of the resource selected for executing a job using ant-algorithm proposed in (Xu et al., 2003) and the possibility of the resource with the highest possibility is less than a certain threshold, then the job will be scheduled to the resource selected by the ant-algorithm (Xu et al., 2003). Otherwise, the scheduler selects another resource and the above procedure will be repeated. The selection of the threshold plays an important role.

Since this modified algorithm takes the resources with highest possibility into consideration, although the processing time is reduced, the processing cost of the jobs may increase when is compared to ant-algorithm (Xu et al., 2003). The inclusion of price factor into this modified algorithm minimizes the total execution time as well as the processing cost of the jobs. The price factor is selected by the Grid user who submits the jobs. For the experiments, the authors have been selected *1/number_of_resources* as the threshold value. In order to evaluate the proposed job scheduler GridSim toolkit (Buyya and Murshed, 2002) have been used.

ANALYSIS OF ACO-BASED SCHEDULING APPROACHES

This section presents a summary of the schedulers based on Ant Colony Optimization described in the previous section. Table 1 summarizes each one of the objectives –among other aspects– for which these algorithms have been applied. Specifically, each one of the columns in the Table 1 is described below:

- **Paper:** Contains a reference to the paper in which the authors describe the proposed work.
- **Distributed Paradigm:** Is the kind of distributed environment in which the different authors have applied their scheduling techniques. "Grid" is an abbreviation for "Grid Computing" and "Cloud" refers to "Cloud Computing".
- **Additional Technique:** Indicates whether the authors have combined the ACO technique with another metaheuristic technique.
- **Objectives:** List the objectives to be minimized or maximized by the proposed ACO algorithm.
- **Algorithm Evaluation:** Refers to the type of environment in which the experiments were performed by the authors. The environment may be for example a real platform, a simulated environment (when details about the simulation tools are not given in the articles), or a specific simulation toolkit.
- **Values of Input Variables:** Here are the values of the ACO-specific variables that were used in the algorithms. Possible variables are:
  - α, importance of trail intensity.
  - β, importance of resource.
  - ρ, permanence of pheromone trail.
  - $(1-\rho)$, evaporation of pheromone trail.
  - P, overhead incurred in resource.

  - τ0, initial pheromone.
  - ce, encouragement coefficient.
  - cp, punishment coefficient.
  - c, coefficient of load balancing factor.
  - ants, the number of ants in the colony, i.e. the number of ants created by the ACO algorithm and sent to find the most suitable machines.
  - Cost per sec, the processing cost per second of machines.
  - Mutation rate, a probability used to decide if an ant moves to a random machine.
  - Decay rate, a factor that causes the mutation rate decreases as time passes.
  - --- means that it does not apply to the category or the authors did not provide information.
- **Experiment Size:** Describes the number of jobs and the number of machines used in the performed experiments.
- **Extra Output:** Tells whether the authors have measured other metrics in the experiments in addition to the ones strictly associated to the objectives.
- **Resource Allocation:** Refers to the time in which the allocation of jobs to resources is scheduled. Static resource allocation means that when the allocation of jobs to resources takes place, the scheduler has complete information in advance, i.e. the scheduler knows the details of both the jobs to allocate and the available resources. On the other hand, dynamic resource allocation means that the jobs to be scheduled arrive at different times, and moreover that resource availability changes over time. A hybrid resource allocation is when some of the jobs are known in advance, and other jobs arrive at different times to be scheduled for execution and details of them are not available until they are received.

As can be seen in algorithms included in Table 1, pheromone trail modeling has been subject of great attention in order to improve the schedulers and to achieve the proposed objectives. A modification of this type is for example proposed in (Hui et al., 2005; Ruay-Shiung et al., 2009). Here, the authors add a load balancing factor to the pheromone trail. With this factor resources have similar completion rates, and thus the ability of load balancing across the overall system is improved. Many authors (Lorpunmanee et al., 2007; Mathiyalagan et al., 2010; Banerjee et al., 2009; Srinivasan et al., 2005; Palmieri and Castagna, 2007) have taken into account in the pheromone update rules the use of a pheromone evaporation rate and pheromone permanence rate. The pheromone evaporation rate is used to indicate that some of the paths traveled by ants are not very interesting, and to prevent that other ants choose those paths. On the other hand, the pheromone permanence rate is used to strengthen the most interesting paths, i.e. the paths chosen by the largest number of ants.

Specifically, the BACO algorithm proposed in (Ruay-Shiung et al., 2009) changes the pheromone update rules (local and global) to achieve better load balancing in the system. The local update rule refreshes the status of a selected resource after job allocation. On the other hand, the global update rule updates the status of each resource for all jobs after completion of each job. Thus, the scheduler keeps updated information of all resources for the next resource allocation round.

Moreover, in (Zehua and Xuejie, 2010; Kousalya and Balasubramanie, 2009; Ritchie and Levine, 2004) the authors have combined the classical ACO algorithm with other techniques or algorithms, such as Local Search or Tabu search. These algorithms have been helpful for researchers to obtain better results than classical (raw) SI approaches. However, just by looking at the Table 1, it clearly seems that the community is still influenced by the idea of developing "pure" ACO-based approaches rather than using complementary metaheuristics.

Table 1. Summary of swarm intelligence approaches

Paper	Distributed paradigm	Additional technique	Objectives	Algorithm evaluation	Values of input variables	Experiment size	Extra outputs	Resource allocation
Lorpunmanee et al., 2007	Grid	---	Minimize makespan	GridSim toolkit	ants=30, β=0.5, ρ=0.9,	3000 jobs 10-20 machines	Total scheduling time	Dynamic
Hui et al., 2005	Grid	---	Maximize load balancing	Simulated Grid	α=0.5, β=0.5, P=0.99, ce=0.0003, cp=0.002, c=4	1000 jobs 10 machines	Simulating time	Dynamic
Mathiyalagan et al., 2010	Grid	---	Minimize makespan	GridSim toolkit	---	10-20 jobs 5 machines	---	Static
Fidanova and Durchova, 2005	Grid	---	Maximize load balancing	Simulated Grid	τ0=0.01, ρ=0.5, ants=1	20 jobs 5 machines	---	Hybrid
Ruay-Shiung et al., 2009	Grid	---	Minimize makespan. Maximize load balancing	Real Grid (Taiwan UniGrid platform + GT4 Globus toolkit)	α=0.5, β=0.5, P=0.99, ce=0.0003, cp=0.002, c=0.4	1000 jobs 25 machines	Average execution time per job and standard deviation of load	Dynamic
Zehua and Xuejie, 2010	Cloud	Complex network	Maximize load balancing	Ad-hoc complex network with Java algorithm simulation software	---	1000 jobs 100 machines	Standard deviation of load	Dynamic
Banerjee et al., 2009	Cloud	---	Minimize makespan	Real Cloud (Google App Engine + Microsoft Live Mesh)	τ0=0.01, ρ=0.5, one ant per service	25 services request 5 machines	---	Dynamic
Xu et al., 2003	Grid	---	Minimize makespan. Maximize load balancing	Simulated Grid	α=0.5, β=0.5, ρ=0.8, ce=1.1, cp=0.8	20 jobs 10 machines	Resource average usage ratio	Dynamic
Srinivasan et al., 2005	Grid	---	Minimize makespan. Minimize monetary cost	Simulated Grid	ants=1-4	---	Allocation cost	Dynamic
Kousalya and Balasubramanie, 2009	Grid	Local search	Minimize makespan. Maximize load balancing	Simulated Grid	---	512 jobs 16 machines	---	Dynamic
Ritchie and Levine, 2004	Grid	Local and tabu search	Minimize makespan	Simulated Grid	α=1-50, β=1-50, τ0=0.01, ρ=0.75, ants=10	512 jobs 16 machines	---	Static

continued on following page

Table 1. Continued

Paper	Distributed paradigm	Additional technique	Objectives	Algorithm evaluation	Values of input variables	Experiment size	Extra outputs	Resource allocation
Sathish and Reddy, 2008	Grid	---	Minimize makespan. Maximize load balancing	GridSim toolkit	Cost per sec=1-7 Threshold=0.1-0.5	40 jobs 20 machines	---	Dynamic
Palmieri and Castagna, 2007	Grid	---	Minimize makespan. Maximize load balancing	Simulated Grid	α=15, β=10, ρ=0.8	512-4096 jobs 32-256 machines	Simulation time	Dynamic
Ludwig and Moallem, 2011	Grid	---	Minimize makespan. Minimize monetary cost. Maximize load balancing	GridSim toolkit	Mutation rate=0.5 Decay rate=0.2	1000 jobs 100 machines	Simulation time	Dynamic

Another fact that can be observed is that most works have been validated in simulated environments, with a number of used jobs that do not exceed 1000 jobs. Within the Distributed Computing community, it is broadly accepted to establish simulated experimental scenarios due to the inherent difficulty of performing tests in real environments. Nevertheless, in real scientific experiments, such as those described at the beginning of this work, the number of jobs to be performed can far exceed that amount. It is then necessary to consider how these algorithms based on ACO respond to situations of greater stress on the machines, at least in simulated scenarios.

A fundamental issue to achieve High Performance in Distributed Computing systems is resource allocation. As the reader can see from Table 1, the authors have proposed different resource allocations techniques. Resource allocation is used to assign the available resources in an efficient way. This means to schedule jobs on the resources required by those jobs while taking into consideration both the resource availability and the size of jobs to run. Distributed job scheduling is broadly classified according to resource allocation as static and dynamic. On one hand, static schemes use a priori knowledge about jobs behavior and do not obtain information about dynamically changes of the environment, i.e. available resources. On the other hand, dynamic schemes make few assumptions about jobs characteristics and obtain information about the jobs and available resources before making a job scheduling decision.

As shown in the Table 1, some researchers (Mathiyalagan et al., 2010; Ritchie and Levine, 2004) have proposed job scheduling algorithms that use static resource allocation. Due to the fact that both Grid Computing and Cloud Computing are environments where the availability of resources is highly dynamic by nature and the jobs arrive over time, applying static resource allocation is unrealistic in practice. On the other hand, in real systems available today there is great difficulty of obtaining complete information about the jobs and resources in advance. In this sense, hybrid resource allocation schemes have been also proposed in an attempt to provide a balance to this trade-off.

On the upside, each one of the changes introduced by the authors to the original ACO algorithm has made the resulting algorithms more efficient according to their objectives. All modifications have achieved better schedulers to complete the execution of jobs within a minimum time and use the resources more efficiently. The Table 1 evidences, however, that most of these algorithms

are focused on minimizing makespan and achieve a proper load balancing, but they do not deal with other interesting and important metrics such as energy use.

Precisely, the development of computing systems has been traditionally focused on performance improvements, for example minimizing execution times of the applications. Due to the large growth of scientific applications, more and more resources are necessary for processing. As a consequence, energy consumption has become a crucial problem (Liu and Zhu, 2010; Baliga et al., 2011; Beloglazov et al., 2011), on one hand because it has started to limit further performance growth due to expensive electricity bills, and on the other hand, by the environmental impact in terms of carbon dioxide (CO2) emissions caused by high energy consumption. This problem arises due to the processing of large amounts of data, management and switching of communications, and so on, and in fact has gave birth to a new field called Green Computing (Li and Zhou, 2011).

In distributed environments such as Cloud Computing it is important minimize energy consumption within data centers. It is also important to consider the energy required to transport data to and from the end-user and the energy consumed in the process of doing so. We believe that since job scheduling techniques based on ACO have proven to be highly efficient in optimization problems can also be good candidates to address the problem mentioned above. Energy consumption could be then a variable within the objectives of a new algorithm. Indeed, an exhaustive search for journal papers regarding SI-based job schedulers for Cloud Computing that do consider energy consumption –which was performed at the time of writing this paper– yielded as a result only one paper (Jeyarani et al., 2011). This shows the undeveloped nature of the topic and therefore the research opportunities therein.

Finally, a distinctive feature of the surveyed works not shown in the Table 1 is that they do not consider job priority. Particularly, for running PSEs, this is a very important aspect. For example, when designing a PSE as N sets of jobs, where every job in a set *p* is associated a particular value for the *ith* variable of the model being simulated by the PSE, job running times between sets can be very different. This is due to the fact that running the same PSE code or solver (i.e. job) varies according to the variable being tested. Sometimes important variations may occur between jobs in the same set as well. These situations are very undesirable since the user can not process/visualize the outputs until *all* jobs finish. Therefore, giving higher priority to jobs that are supposed to take longer to finish may help in reducing makespan and hence improve output processing.

CONCLUSION AND FUTURE RESEARCH DIRECTIONS

Parameter Sweep Experiments (PSE) is a type of numerical simulations that involves a large number of independent jobs and requires a lot of computing power. These jobs must be efficiently processed in the different computing resources of a distributed environment such as Grid Computing or Cloud Computing. Here, job scheduling becomes crucial.

To solve this problem, many researchers have proposed a large number of schedulers based on Ant Colony Optimization. In this work we have summarized different approaches in Table 1. As the reader can see, we have described problems of job scheduling where different authors have made several changes to the metaheuristics to achieve different goals, i.e. minimize the total execution time of jobs, minimize cost, and maximize load balancing or some combination thereof. In addition, the surveyed algorithms have been run or simulated on two types of distributed environments, i.e. Grid Computing and Cloud Computing. The simulation environments have been used to assess the algorithms performance and effectiveness for job scheduling.

An important issue to note is that most of the papers are aimed at Grid Computing environments, while very few are designed for Cloud Computing, which is due to the fact that Cloud Computing is a paradigm more recent than Grid Computing and much less popular among scientists and engineers. Cloud Computing, interestingly, is a paradigm that offers the means for building the next generation parallel computing infrastructures along with ease of use. Although the use of Clouds finds its roots in IT environments, the idea is gradually entering scientific and academic ones. Even when Cloud Computing is popular, little research has been done with respect to evaluating the benefits of the paradigm for scheduling and executing resource intensive scientific applications. It is also important to note that to date the algorithms designed to address job scheduling for Clouds come from adaptations of job schedulers for Grids. The reason is that researchers have greater knowledge of the latter (Kaur, 2011).

Since currently there is a lack of profound studies in the literature about the viability of using Cloud Computing to execute scientific and engineering applications from a performance standpoint, we have presented in a previous work (Pacini et al., 2011) an empirical study on the employment of Cloud infrastructures to run PSEs using scheduling policies commonly used in Clouds (i.e. time-shared and space-shared)**.** The results showed when running PSEs in Clouds, near-to-ideal speedups can be obtained. We believe this line of research could greatly benefit from SI-based scheduling approaches to optimize other aspects other than speedup.

We are extending this work in several directions. First, we are surveying other types of Swarm Intelligence algorithms to job scheduling in distributed environment such as Particle Swarm Optimization (Kennedy and Eberhart, 1995; Kennedy and Eberhart, 2001), Artificial Fish Swarm Algorithm (Li et al., 2002) and Bee Colony Optimization (Lucic and Teodorovic, 2003). We think this will not dramatically reshape the comparison framework depicted in Table 1, however it will certainly enhance our analysis. Second, due to the fact that currently there are few efforts devoted to job scheduling in Clouds, we aim at designing a new SI-based scheduler that is capable of efficiently running PSEs in Cloud Computing environments while addressing important aspects such as energy consumption and PSE job priorities. We are also planning to embed the resulting scheduler into CloudSim (Calheiros et al., 2011) in order to provide empirical evidence of its effectiveness. Eventually, we will implement the scheduler on top of a real (not simulated) Cloud platform, such as Eucalyptus1, OpenNebula 2and Emotive Cloud3.

Finally, an interesting research line has recently arisen as a consequence of the astonishingly and increasing number of available mobile devices such as smartphones. Nowadays, mobile devices have a remarkable amount of computational resources that allows them to execute complex applications, such as 3D games, and to store large amounts of data. In fact, recent work has experimentally shown the feasibility of using such devices for executing computing intensive scientific codes (Rodriguez et al., 2011). Due to these advances, emergent research lines have aimed at integrating smartphones and other kind of mobile devices into traditional distributed computational environments, such as clusters and Grids (Rodriguez et al., 2011b). However, intuitively, job scheduling in these highly heterogeneous environments is more challenging since mobile devices rely on unreliable wireless connections and batteries, which is necessary to consider at the scheduling level. This will on the other hand provide excellent research opportunities for new schedulers based on traditional optimization techniques as well as SI-based ones.

REFERENCES

Alabas-Uslu, C., & Dengiz, B. (2011). A self-adaptive local search algorithm for the classical vehicle routing problem. *Expert Systems with Applications*, *38*(7), 8990–8998. doi:10.1016/j.eswa.2011.01.116

Axelrod, R. (1997). The dissemination of culture: A model with local convergence and global polarization. *The Journal of Conflict Resolution*, *41*(2), 203–226. doi:10.1177/0022002797041002001

Baker, M., Buyya, R., & Laforenza, D. (2002). Grids and grid technologies for wide-area distributed computing. *Software, Practice & Experience*, *32*, 1437–1466. doi:10.1002/spe.488

Baliga, J., Ayre, R., Hinton, K., & Tucker, R. (2011). Green cloud computing: Balancing energy in processing, storage and transport. *Proceedings of the IEEE*, *99*(1), 149–167. doi:10.1109/JPROC.2010.2060451

Banerjee, S., Mukherjee, I., & Mahanti, P. (2009). Cloud computing initiative using modified ant colony framework. In *World Academy of Science, Engineering and Technology*, (pp. 221–224). WASET.

Basney, J., Livny, M., & Mazzanti, P. (2000). *Harnessing the capacity of computational grids for high energy physics*. In Conference on Computing in High Energy and Nuclear Physics.

Baterina, A., & Oppus, C. (2010). Image edge detection using ant colony optimization. *WSEAS Transactions in Signal Processing*, *6*(2), 58–67.

Beloglazov, A., Buyya, R., Lee, Y., & Zomaya, A. (2011). A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, *82*, 47–111. doi:10.1016/B978-0-12-385512-1.00003-7

Beni, G., & Wang, J. (1989). *Swarm intelligence in cellular robotic systems*. In NATO Advanced Workshop on Robotics and Biological Systems.

Berglund, A. C., Elmroth, E., Hernandez, F., Sandman, B., & Tordsson, J. (2009). *Combining local and grid resources in scientific workflows (for bioinformatics)*. In The 9th International Workshop on State-of-the-Art in Scientific and Parallel Computing, Lecture Notes in Computer Science. Springer-Verlag.

Bianchi, L., Gambardella, L., & Dorigo, M. (2002). An ant colony optimization approach to the probabilistic traveling salesman problem. In *Parallel Problem Solving from Nature - PPSN VII*. In *Lecture Notes in Computer Science* (*Vol. 2439*, pp. 883–892). Berlin, Germany: Springer.

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: From natural to artificial systems*. Oxford University Press.

Braun, T., Siegel, H., Beck, N., Bölöni, L., Maheswaran, M., & Reuther, A. (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, *61*(6), 810–837. doi:10.1006/jpdc.2000.1714

Buyya, R., & Murshed, M. (2002). Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurrency and Computation [CCPE]. *Practice and Experience*, *14*(13), 1175–1220. doi:10.1002/cpe.710

Buyya, R., Yeo, C., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, *25*(6), 599–616. doi:10.1016/j.future.2008.12.001

Calheiros, R., Ranjan, R., Beloglazov, A., De Rose, C., & Buyya, R. (2011). CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software, Practice & Experience*, *41*(1), 23–50. doi:10.1002/spe.995

Careglio, C., Monge, D., Pacini, E., Mateos, C., Mirasso, A., & Garino, C. G. (2010). Sensibilidad de resultados del ensayo de tracción simple frente a diferentes tamaños y tipos de imperfecciones. *Mecánica Computacional*, *29*(41), 4181–4197.

Carpi, L., Rosso, O., Saco, P., & Ravetti, M. G. (2011). Analyzing complex networks evolution through information theory quantifiers. *Physics Letters. [Part A]*, *375*(4), 801–804. doi:10.1016/j.physleta.2010.12.038

Catlett, C. (2005). Teragrid: A foundation for us cyberinfrastructure. In *NPC´ 05, Network and Parallel Computing*. Berlin, Germany: Springer. doi:10.1007/11577188_1

Chandra, B., & Baskaran, R. (2012). A survey: Ant colony optimization based recent research and implementation on several engineering domain. *Expert Systems with Applications*, *39*(4), 4618–4627. doi:10.1016/j.eswa.2011.09.076

Coveney, P., Chin, J., Harvey, M., & Jha, S. (2005). Scientific grid computing: The first generation. *Computing in Science & Engineering*, *7*, 24–32. doi:10.1109/MCSE.2005.100

Den Besten, M., Stutzle, T., & Dorigo, M. (2000). Ant colony optimization for the total weighted tardiness problem. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo, & H. Schwefel (Eds.), *Parallel Problem Solving from Nature PPSN VI, volume 1917 of Lecture Notes in Computer Science,* (pp. 611–620). Berlin, Germany: Springer.

Dorigo, M. (1992). *Optimization, learning and natural algorithms*. PhD thesis, Politecnico di Milano, Italy.

Dorigo, M., & Caro, G. D. (1999). *The ant colony optimization metaheuristic*. Maidenhead, UK: McGraw-Hill Ltd.

Dorigo, M., & Stützle, T. (2003). The ant colony optimization metaheuristic: Algorithms, applications, and advances. In F. Glover & G. Kochenberger (Eds.), *Handbook of metaheuristics, volume 57 of international series in operations research & management science*, (pp. 250–285). New York, NY: Springer.

Etminani, K., & Naghibzadeh, M. (2007). A min-min max-min selective algorithm for grid task scheduling. In *3rd International Conference in Central Asia*, (pp. 1–7). IEEE Computer Society/IFIP.

Fidanova, S., & Durchova, M. (2005). Ant algorithm for Grid scheduling problem. In *5th International Conference on Large-Scale Scientific Computing*, (pp. 405–412). Springer.

Foster, I., & Iamnitchi, A. (2003). On death, taxes, and the convergence of peer-to-peer and grid computing. In *2nd International Workshop on Peer-to-Peer Systems* (IPTPS 2003), (pp. 118–128). Berkeley, CA.

Foster, I., & Kesselman, C. (2003). *The grid: Blueprint for a new computing infrastructure*. San Francisco, CA: Morgan Kaufmann Inc.

Foster, I., Kesselman, C., & Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *The International Journal of Supercomputer Applications*, *15*(3), 200–222. doi:10.1177/109434200101500302

Gagliardi, F., & Begin, M. (2005). Egee - Providing a production quality grid for e-science. In *Proceedings of the 2005 IEEE International Symposium on Mass Storage Systems and Technology*, (pp. 88–92). Washington, DC: IEEE Computer Society.

García Martínez, C., Cordón, O., & Herrera, F. (2007). A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria tsp. *European Journal of Operational Research*, *180*(1), 116–148. doi:10.1016/j.ejor.2006.03.041

Garrigues, C., Robles, S., Borrell, J., & Navarro-Arribas, G. (2010). Promoting the development of secure mobile agent applications. *Journal of Systems and Software*, *83*(6), 959–971. doi:10.1016/j.jss.2009.11.001

Gropp, W., Lusk, E., & Skjellum, A. (1994). *Using MPI: Portable parallel programming with the message passing interface*. MIT Press.

Gulamali, M., Mcgough, A., Newhouse, S., & Darlington, J. (2004). *Using ICENI to run parameter sweep applications across multiple Grid resources*. In Global Grid Forum 10, Case Studies on Grid Applications Workshop.

Hao, Y., Zhang, Y., & Cao, J. (2010). Web services discovery and rank: An information retrieval approach. *Future Generation Computer Systems*, *26*(8), 1053–1062. doi:10.1016/j.future.2010.04.012

Heinonen, J., & Pettersson, F. (2007). Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem. *Applied Mathematics and Computation*, *187*(2), 989–998. doi:10.1016/j.amc.2006.09.023

Huang, G., & Wang, Q. (2011). A hybrid aco-ga on sports competition scheduling. In Ostfeld, A. (Ed.), *Ant colony optimization - Methods and applications* (pp. 89–100). InTech.

Huang, K., & Liao, C. (2008). Ant colony optimization combined with taboo search for the job shop scheduling problem. *Computers & Operations Research*, *35*(4), 1030–1046. doi:10.1016/j.cor.2006.07.003

Hui, Y., Xue-Qin, S., Xing, L., & Ming-Hui, W. (2005). An improved ant algorithm for job scheduling in Grid computing. In *International Conference on Machine Learning and Cybernetics*, Vol. 5, (pp. 2957–2961). IEEE Computer Society.

Jeyarani, R., Nagaveni, N., & Vasanth Ram, R. (2011). Design and implementation of adaptive power-aware virtual machine provisioner (APA-VMP) using swarm intelligence. *Future Generation Computer Systems*, *28*(5).

Kaur, P., & Chana, I. (2011). Enhancing grid resource scheduling algorithms for cloud environments. In A. Mantri, S. Nandi, G. Kumar, & S. Kumar (Eds.), *High Performance Architecture and Grid Computing, volume 169 of Communications in Computer and Information Science*, (pp. 140–144). Berlin, Germany: Springer.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *IEEE International Conference on Neural Networks*, Vol. 4, (pp. 1942–1948). IEEE Computer Society.

Kennedy, J., & Eberhart, R. (2001). *Swarm intelligence*. San Francisco, CA: Morgan Kaufmann Publishers Inc.

Kousalya, K., & Balasubramanie, P. (2007). *Resource scheduling in computational grid using ant algorithm.* In International Conference on Computer Control and Communications, Pakistan.

Kousalya, K., & Balasubramanie, P. (2008). An enhanced ant algorithm for grid scheduling problem. *IJCSNS International Journal of Computer Science and Network Security*, *8*(4), 262–271.

Kousalya, K., & Balasubramanie, P. (2009). To improve ant algorithm's grid scheduling using local search. *International Journal of Intelligent Information Technology Application*, *2*(2), 71–79.

Kuo, I., Horng, S., Kao, T., Lin, T., Lee, C., & Chen, Y. (2010). A hybrid swarm intelligence algorithm for the travelling salesman problem. *Expert Systems: International Journal of Knowledge Engineering and Neural Networks*, *27*(10), 166–179. doi:10.1111/j.1468-0394.2010.00517.x

Levine, J., & Ducatelle, F. (2004). Ant colony optimization and local search for bin packing and cutting stock problems. *The Journal of the Operational Research Society*, *55*, 705–716. doi:10.1057/palgrave.jors.2601771

Li, Q., & Zhou, M. (2011). The survey and future evolution of green computing. In 2011 IEEE/ACM International Conference on Green Computing and Communications (Green-Com), (pp. 230–233). IEEE.

Li, X., Shao, Z., & Qian, J. (2002). An optimizing method based on autonomous animals: Fish-swarm algorithm. *Systems Engineering Theory and Practice*, *22*(11), 32–38.

Liao, L., & Huang, C. (2011). Tabu search heuristic for two-machine flowshop with batch processing machines. *Computers & Industrial Engineering*, *60*(3), 426–432. doi:10.1016/j.cie.2010.03.004

Liu, Y., & Zhu, H. (2010). A survey of the research on power management techniques for high-performance systems. *Software, Practice & Experience*, *40*(11), 943–964. doi:10.1002/spe.952

Lorpunmanee, S., Sap, M., Abdullah, A., & Chompooinwai, C. (2007). *An ant colony optimization for dynamic job scheduling in Grid environment* (pp. 314–321). World Academy of Science, Engineering and Technology.

Lucic, P., & Teodorovic, D. (2003). Computing with bees: Attacking complex transportation engineering problems. *International Journal of Artificial Intelligence Tools*, *12*(3), 375–394. doi:10.1142/S0218213003001289

Ludwig, S., & Moallem, A. (2011). Swarm intelligence approaches for grid load balancing. *Journal of Grid Computing*, *9*(3), 279–301. doi:10.1007/s10723-011-9180-5

Ma, T., & Buyya, R. (2005). Critical-path and priority based algorithms for scheduling workflows with parameter sweep tasks on global grids. In *Proceedings of the 17th International Symposium on Computer Architecture on High Performance Computing*, Vol. 8, (pp. 251–258). Washington, DC: IEEE Computer Society.

Mathiyalagan, P., Suriya, S., & Sivan, S. (2010). Modified ant colony algorithm for Grid scheduling. *International Journal on Computer Science and Engineering*, *2*, 132–139.

Pacini, E., Ribero, M., Mateos, C., Mirasso, A., & Garino, C. G. (2011). Simulation on cloud computing infrastructures of parametric studies of nonlinear solids problems. In F. Cipolla-Ficarra et al., (Eds.), *Advances in New Technologies, Interactive Interfaces and Communicability* (ADNTIIC 2011), (pp. 56–68). Huerta Grande, Córdoba, Argentina: Blue Herons.

Palmieri, F., & Castagna, D. (2007). Swarm-based distributed job scheduling in next-generation grids. In Elleithy, K. (Ed.), *Advances and innovations in systems, computing sciences and software engineering* (pp. 137–143). Netherlands: Springer. doi:10.1007/978-1-4020-6264-3_25

Pordes, R., Petravick, D., Kramer, B., Olson, D., Livny, M., Roy, A., … Quick, R. (2007). The open science grid. *Journal of Physics: Conference Series, 78*(1), 012–057.

Ritchie, G., & Levine, J. (2004). A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments. In *Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group.*

Rizzoli, A., Oliverio, F., Montemanni, R., & Gambardella, M. (2004). Ant colony optimisation for vehicle routing problems: From theory to applications. *Galleria Rassegna Bimestrale Di Cultura*, *9*(1), 1–50.

Rodriguez, J. M., Mateos, C., & Zunino, A. (2011). Are smartphones really useful for scientific computing? In F. V. Cipolla-Ficarra et al., (Eds.), *Advances in New Technologies, Interactive Interfaces and Communicability (ADNTIIC 2011), Lecture Notes in Computer Science*, (pp. 35–44). Huerta Grande, Córdoba, Argentina, December 2011.

Rodriguez, J. M., Zunino, A., & Campo, M. (2011). Introducing mobile devices into Grid systems: A survey. *International Journal of Web and Grid Services*, *7*(1). doi:10.1504/IJWGS.2011.038386

Ruay-Shiung, C., Jih-Sheng, C., & Po-Sheng, L. (2009). An ant algorithm for balanced job scheduling in grids. *Future Generation Computer Systems*, *25*, 20–27. doi:10.1016/j.future.2008.06.004

Samples, M., Daida, J., Byom, M., & Pizzimenti, M. (2005). Parameter sweeps for exploring GP parameters. *In Conference on Genetic and Evolutionary Computation, GECCO '05*, (pp. 212–219). New York, NY: ACM Press.

Sathish, K., & Reddy, A. R. M. (2008). Enhanced ant algorithm based load balanced task scheduling in grid computing. *IJCSNS International Journal of Computer Science and Network Security*, *8*(10), 219–223.

Srinivasan, T., Siddharth, J., Jayesh, S., & Chandrasekhar, A. (2005). A comprehensive architectural framework for task management in scalable computational grids. In *Annual National level Technical Symposium conducted by the Computer Science and Engineering Association of the Department of Computer Science and Engineering* (DCSE), (pp. 14–15). Chennai, India: College of Engineering, Anna University.

Strogatz, S. (2001). Exploring complex networks. *Nature*, *410*(6825), 268–276. doi:10.1038/35065725

Stützle, T., & Hoos, H. (2000). Max-min ant system. *Future Generation Computer Systems*, *16*(8), 889–914.

Sun, C., Kim, B., Yi, G., & Park, H. (2004). A model of problem solving environment for integrated bioinformatics solution on grid by using condor. In *Grid and Cooperative Computing*, (pp. 935–938).

Taylor, I., Deelman, E., Gannon, D., & Shields, M. (2006). *Workflows for e-science: Scientific workflows for grids*. New York, NY: Springer-Verlag Inc.

Thain, D., Tannenbaum, T., & Livny, M. (2005). Distributed computing in practice: The Condor experience. *Concurrency and Computation*, *17*(2-4), 323–356. doi:10.1002/cpe.938

Vecchiola, C., Pandey, S., & Buyya, R. (2009). High-performance cloud computing: A view of scientific applications. In *Proceedings of the 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, ISPAN '09, (pp. 4–16). Washington, DC: IEEE Computer Society.

Vesel, A., & Zerovnik, J. (2000). How good can ants color graphs? *Journal of Computing and Information Technology*, *8*, 131–136. doi:10.2498/cit.2000.02.04

Wang, L., Tao, J., Kunze, M., Castellanos, A. C., Kramer, D., & Karl, W. (2008). Scientific cloud computing: Early definition and experience. In *10th IEEE International Conference on High Performance Computing and Communications*, (pp. 825–830). Washington, DC: IEEE Computer Society.

Woeginger, G. (2003). Exact algorithms for np-hard problems: A survey. In M. Junger, G. Reinelt, & G. Rinaldi (Eds.), *Combinatorial Optimization - Eureka, You Shrink! volume 2570 of Lecture Notes in Computer Science,* (pp. 185–207). Berlin, Germany: Springer.

Wozniak, J., Striegel, A., Salyers, D., & Izaguirre, J. (2005). GIPSE: Streamlining the management of simulation on the Grid. In *38th Annual Simulation Symposium*, (pp. 130–137).

Xu, Z., Hou, X., & Jizhou, S. (2003). Ant algorithm-based task scheduling in grid computing. In *CCECE 2003 - CCGEl 2003*, (pp. 1107–1110). Montreal.

Yawei, L., & Zhiling, L. (2004). *A survey of load balancing in grid computing* (pp. 280–285). CIS.

Youn, C., & Kaiser, T. (2010). Management of a parameter sweep for scientific applications on cluster environments. *Concurrency and Computation*, *22*, 2381–2400. doi:10.1002/cpe.1563

Yun-Chia, L., & Smith, A. (2004). An ant colony optimization algorithm for the redundancy allocation problem (RAP). *IEEE Transactions on Reliability*, *53*(3), 417–423. doi:10.1109/TR.2004.832816

Zehua, Z., & Xuejie, Z. (2010). A load balancing mechanism based on ant colony and complex network theory in open Cloud Computing federation. In *2nd International Conference on Industrial Mechatronics and Automation*, (pp. 240–243). IEEE Computer Society.

Zhang, Q., & Theodoropoulos, G. (2003). Towards an asynchronous MIPS processor. In A. Omondi & S. Sedukhin, (Eds.), *Advances in Computer Systems Architecture, volume 2823 of Lecture Notes in Computer Science,* (pp. 137–150). Berlin, Germany: Springer.

ADDITIONAL READING

Afshar, M. (2010). A parameter free continuous ant colony optimization algorithm for the optimal design of storm sewer networks: Constrained and unconstrained approach. *Advances in Engineering Software*, *41*(2), 188–195. doi:10.1016/j.advengsoft.2009.09.009

AlRashidi, M., & El-Hawary, M. (2009). A survey of particle swarm optimization applications in electric power systems. *IEEE Transactions on Evolutionary Computation*, *13*(4), 913–918. doi:10.1109/TEVC.2006.880326

Biswal, B., Dash, P., & Mishra, S. (2011). A hybrid ant colony optimization technique for power signal pattern classification. *Expert Systems with Applications*, *38*(5), 6368–6375. doi:10.1016/j.eswa.2010.11.102

Bonabeau, E., Corne, D., & Poli, R. (2010). Swarm intelligence: The state of the art special issue of natural computing. *Natural Computing*, *9*(3), 655–657. doi:10.1007/s11047-009-9172-6

Cai, Y. (2010). Artificial fish school algorithm applied in a combinatorial optimization problem. [IJISA]. *International Journal of Intelligent Systems and Applications*, *2*(1), 37–43. doi:10.5815/ijisa.2010.01.06

Chandra-Mohan, B., & Baskaran, R. (2012). A survey: Ant colony optimization based recent research and implementation on several engineering domain. *Expert Systems with Applications*, *39*(4), 4618–4627. doi:10.1016/j.eswa.2011.09.076

Ciruela Martín, S. (2008) La sabiduría de la naturaleza. *Ciencia Cognitiva – Revista Electrónica de Divulgación, 2*, 81–83.

Díaz, J., Reyes, S., Badia, R., & No, A. N., & noz Caro, C. M. (2010). A general model for the generation and scheduling of parameter sweep experiments in computational grid environments. *Procedia Computer Science*, *1*(1), 565–572. doi:10.1016/j.procs.2010.04.060

Dougherty, B., White, J., & Schmidt, D. (2012). Model-driven auto-scaling of green cloud computing infrastructure. *Future Generation Computer Systems*, *28*(2), 371–378. doi:10.1016/j.future.2011.05.009

Foster, I., Yong, Z., Raicu, I., & Lu, S. (2008). Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop*, (pp. 1–10). IEEE.

Huang, Y., Bessis, N., Norrington, P., Kuonen, P., & Hirsbrunner, B. (2011). (in press). Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm. *Future Generation Computer Systems*. doi:10.1016/j.future.2011.05.006

Kameyama, K. (2009). Particle swarm optimization - A survey. *IEICE Transactions on Information and Systems, E92.D*(7), 1354–1361.

Karaboga, D., & Akay, B. (2009). A survey: algorithms simulating bee swarm intelligence. *Artificial Intelligence Review*, *31*(1-4), 61–85. doi:10.1007/s10462-009-9127-4

Kolias, C., Kambourakis, G., & Maragoudakis, M. (2011). Swarm intelligence in intrusion detection: A survey. *Computers & Security*, *30*(8), 625–642. doi:10.1016/j.cose.2011.08.009

Lee, Y., Leu, S., & Chang, R. (2011). Improving job scheduling algorithms in a grid environment. *Future Generation Computer Systems*, *27*(8), 991–998. doi:10.1016/j.future.2011.05.014

Mateescu, G., Gentzsch, W., & Ribbens, C. (2011). Hybrid computing – Where HPC meets grid and cloud computing. *Future Generation Computer Systems*, *27*(5), 440–453. doi:10.1016/j.future.2010.11.003

Neri, F., & Cotta, C. (2011). Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation, 2*(1).

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., & Zagorodnov, D. (2009). The eucalyptus open-source cloud-computing system. In F. Cappello, C. Wang, & R. Buyya (Eds.), *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, (pp. 124–131). IEEE Computer Society.

Pedemonte, M., Nesmachnow, S., & Cancela, H. (2011). A survey on parallel ant colony optimization. *Applied Soft Computing*, *11*(8), 5181–5197. doi:10.1016/j.asoc.2011.05.042

Razavi, S., & Jalali-Farahani, F. (2010). Optimization and parameters estimation in petroleum engineering problems using ant colony algorithm. *Journal of Petroleum Science Engineering*, *74*(3-4), 147–153. doi:10.1016/j.petrol.2010.08.009

Rodero-Merino, L., Vaquero, L., Gil, V., Galán, F., Fontán, J., Montero, R., & Llorente, I. (2010). From infrastructure delivery to service management in clouds. *Future Generation Computer Systems*, *26*(8), 1226–1240. doi:10.1016/j.future.2010.02.013

Schwiegelshohn, U., Badia, R., Bubak, M., Danelutto, M., Dustdar, S., & Gagliardi, F. (2010). Perspectives on grid computing. *Future Generation Computer Systems*, *26*(8), 1104–1115. doi:10.1016/j.future.2010.05.010

Smanchat, S., Indrawan, M., Ling, S., Enticott, C., & Abramson, D. (2011). A scheduler based on resource competition for parameter sweep workflow. *Procedia Computer Science*, *4*(0), 176–185. doi:10.1016/j.procs.2011.04.019

Wan, Y., Pei, T., Zhou, C., Jiang, Y., Qu, C., & Qiao, Y. (2012). Acomcd: A multiple cluster detection algorithm based on the spatial scan statistic and ant colony optimization. *Computational Statistics & Data Analysis*, *56*(2), 283–296. doi:10.1016/j.csda.2011.08.001

Wu, S., & Banzhaf, W. (2010). The use of computational intelligence in intrusion detection systems: A review. *Applied Soft Computing*, *10*(1), 1–35. doi:10.1016/j.asoc.2009.06.019

Zhou, A., Qu, B., Li, H., Zhao, S., Suganthan, P., & Zhang, Q. (2011). Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, *1*(1), 32–49. doi:10.1016/j.swevo.2011.03.001

KEY TERMS AND DEFINITIONS

Ant Colony Optimization (ACO): Is a class of optimization algorithms modeled on the actions of an ant colony. ACO methods are useful in problems that need to find paths to goals. Artificial 'ants' locate optimal solutions by moving through a parameter space representing all possible solutions. Real ants lay down pheromones directing each other to resources while exploring their environment. The simulated 'ants' similarly record their positions and the quality of their solutions, so that in later simulation iterations more ants locate better solutions.

Cloud Computing: A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers. Cloud Computing is the delivery of computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a utility over a network (typically the Internet).

Grid Computing: Is a model of distributed computing that uses geographically and administratively disparate resources. A Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed 'autonomous' resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements. Individual users can access computers and data transparently, without having to consider location, operating system, account administration, and other details.

Job Scheduling: Is the process of allocating a set of jobs belonging to an application into available computing resources. The main objective is achieving a high system throughput while matching application needs with the available computing resources.

Load Balancing: Is a computer methodology to distribute workload across multiple computers to achieve optimal resource utilization, maximize throughput, minimize response time, and avoid overload. Load balancing divides the amount of work that a computer has to do between two or more computers so that more work gets done in the same amount of time and, in general, all users get served faster.

Makespan: Is defined as the amount of time, from start to finish for completing a set of jobs, i.e. the maximum completion time of all jobs.

Parameter Sweep: a parameter sweep is a type of experiment in which multiple datapoints are examined by executing an algorithm numerous times with different parameter configurations.

Swarm Intelligence (SI): Is a discipline that deals with natural and artificial systems composed of many individuals that coordinate using decentralized control and self-organization. In particular,

SI focuses on the collective behaviors that result from the local interactions of the individuals with each other and with their environment. Examples of systems studied by swarm intelligence are colonies of ants and termites, schools of fish, flocks of birds, herds of land animals.

ENDNOTES

1. http://www.eucalyptus.com/
2. http://www.opennebula.org
3. http://www.emotivecloud.net