On the Challenge of Allocating Service Based Applications in a Grid Environment

Sharath Babu Musunoori, Geir Horn, Frank Eliassen, and Alia Mourad
SIMULA Research Laboratory, P.O. Box 134, 1325 Lysaker, Norway.
E-mails: {sharath, geirho, frank, mouradal}@simula.no

Abstract—The significant advancements in grid computing provide a promising platform for parallel and distributed systems. However, achieving acceptable application performance in the grid environment remains a difficult engineering challenge. In particular for applications composed of services requiring some quality requirements to be fulfilled in order to satisfy the users needs. There is a growing need to support these applications as they perform unacceptably if platform resources are scarce or if the deployment is not carefully configured and tuned for the anticipated load. The fundamental problem we have at hand is of partitioning application services on to a distributed environment such that all services involved in the composition get sufficient resources to allow them to deliver at least the minimum required quality to be useful for the application. A subsequent problem is how to re-partition (reconfigure) or even adapt the application's service set dynamically when the underlying resources of the grid environment change with time. In this paper we provide an overview of both service partitioning and reconfiguration and different elements that affect the search process and investigate the nature of the problem space. In addition, we also introduce a special type of evolutionary algorithms such as learning and other nature-inspired solution techniques that can handle the dynamic behaviour of such stochastic computational environment.

Keywords: Application Configuration, Application Reconfiguration, Scheduling, Mapping, Partitioning, Adaptation.

I. INTRODUCTION

Today’s grid applications mostly run on computers controlled by a single administrative domain and can as such be viewed as a type of cluster applications. Here the user has complete knowledge about the resources at each node in the cluster, and nodes are not subject to unexpected shutdowns or unforeseen resource prioritization. However, the grid vision extends beyond this current usage to a situation where every computer connected to the Internet can be used as a grid resource. The configuration of a grid application will be much more complex in this environment. The grid nodes will be heterogeneous and will join and leave the network at unpredictable time instances. The owner of a node may at any time decide to withdraw resources from the grid to use them for his/her own good. An application that will maintain its usefulness under these conditions must adapt to the changing environment.

Another dominant trend today is to regard applications as composed of services that may be invoked and executed remotely. A grid node can thus be seen as a capsule holding and executing a subset of an application’s services. Consequently, the problem of configuring an application for a dynamic grid environment is that of stochastically partition its service set onto a set of heterogeneous capsules, each offering a set of resources to the running services. In the physical world there will be connections among the capsules with certain network capacities like bandwidth and transmission delays and packet jitter. On the other hand there is a graph among the services defining the application. The problem is consequently to obtain a feasible partition of the service graph over the capsules such that no capsule is oversubscribed with respect to its resources, and such that all the services achieve the minimum quality level, i.e. resources, necessary for the application to run as intended. The first fundamental problem we address is that of partitioning the service set in to the capsules:

- How to split the application’s service set into the number of grid nodes sub-sets?

Given the partitioning problem, analysing the whole solution space is impossible since an exhaustive search through all possible configurations in the solution space takes impractically too long. The only two options left are either to use a trial and error method or a specific heuristic approach that will make the search process efficient such that a feasible solution is found in acceptable time. However, these methods may not work up to the expectation if the solution space has a stochastic behaviour. Despite the success of various heuristics in solving the combinatorial optimisation problems like the one we discuss here, they do not scale and fail to deliver for large systems since they often need the full control over a system in order to make global decisions. One natural extension to this is to model a system of which several individual entities make appropriate decisions that will lead to a global convergence. Such an extension may create a decentralised environment for decision making where all agents coordinate with each other to generate a global behaviour for the system. Such coordination is often a result of the organisation of individual autonomous agents or modules of the system that leads to a global convergence. However, designing such a heuristic is a very complex task since the stochastic nature of the grid environment inserts randomness in solution space. In this paper we provide an overview of different elements that affect the search process and investigate the nature of the problem space.

In the grid computational setting where different varieties of distributed resources are connected to a shared (unified) computing platform, applications and resources (computing
nodes) enter and leave the system. Such changes in resource availability and application needs make the grid environment dynamic. Therefore any configured application service running on the grid platform cannot depend on specific resources, but need a mechanism to adapt to dynamic changes in resource availability. A subsequent problem we address here in this paper is service reconfiguration:

- How to re-arrange or even adapt the application’s service set dynamically when the underlying resources of the grid environment change with time?

Depending on the nature of the change, the corrective actions may either be for reconfiguring the application or for adapting it. By the term reconfiguration in this paper, we shall mean to address any failure within the current running application service and a set of required corrective actions that either restore (or modify) the current configuration or even find a new one such that the minimum quality or performance goals of the overall application are achieved. We define the term adaptation as an adjustment in the non-functional behaviour of the application service by changing its resource assignment. Thus the application adaptation refers to modifying the current running configuration such that the performance of the application is optimised based on the current resource availability. In addition, we introduce two special type of evolutionary algorithms such as learning and nature-inspired algorithms that provide some basis to design new solution techniques that can handle both complexity of the service partitioning and dynamic behaviour of the grid environment.

II. APPLICATION SERVICE PARTITIONING

Let \( S = \{ S_1, S_2, ..., S_{|S|}\} \) be a set of services involved in an application. That is the application is equivalent with its service set \( S \), where all the services in \( S \) are taken to be primitive (or atomic) that can not further be composed of other services. The useful application experienced by the user is consequently a service graph with corresponding (logical) communication between the services. In order to find a suitable resource configuration for the service composition of the services in \( S \), several details like application characteristics, size of input and output data, available network bandwidth and other computational resource availability have to be considered. Let \( |C| \) be a number of computing nodes, or capsules, of the grid environment, each node \( C_m \) has a set of local resources \( R_m = \{ R_{m,1}, R_{m,2}, ..., R_{m,|R_m|}\} \). Let \( Q_i = \{ Q_{i,1}, Q_{i,2}, ..., Q_{i,|Q_i|}\} \) be a set of quality requirements (e.g., latency, frame rate, etc.) that jointly determine the perceived quality of a service \( S_i \). To obtain a quality \( Q_{i,j} \), the service \( S_i \) requires to consume a set of available resources \( r_{C_m, Q_{i,j}} = \{ R_{C_m,1}, R_{C_m,2}, ..., R_{C_m,|R_{C_m}|}\} \) (e.g., CPU, mem, communication links) of any grid node \( C_m \). The \( r_{C_m, Q_{i,j}} \) shows an association between resources \( R_m \) of a grid node \( C_m \) and a quality requirement \( Q_{i,j} \) of a service \( S_i \) (i.e., \( r_{C_m, Q_{i,j}} \subseteq R_m \)).

Despite the fact that various computational, communication, and other (sensors, cameras) resources contribute to deliver the required levels of application quality, the scheduler is required to understand the application service context, service user requirements and availability of dynamically changing resources. By accounting all these requirements, the goal of service configuration is to partition or map the service set \( S \) on to \( |C| \) grid nodes. Let \( \Pi(t) = \{ S_1, S_2, ..., S_{|S|}\} \) be a partition at any instance of time \( t \) such that \( \bigcup_{i=1}^{|C|} S_i = S \) and \( S_i \cap S_j = \emptyset \) for all pairs \( i \neq j \), and \( S_i \neq \emptyset \) for all \( j \). Given this partition \( \Pi(t) \), any service \( S_i \) will try to obtain a set of required resources, \( r_{C_m, Q_{i,j}} \) of its host \( C_m \) for all quality dimensions \( j \). In other words, \( r_{C_m, S_i} = \{ r_{C_m, Q_{i,1}}, r_{C_m, Q_{i,2}}, ..., r_{C_m, Q_{i,|Q_i|}}\} \) is a set of resources required for a service \( S_i \). Similarly, let the set \( r_{C_m} = \{ r_{C_m, S_i}, r_{C_m, S_j}, ..., r_{C_m, S_{|S|}}\} \) be a set of all possible resource capabilities for all services in \( S \) respectively. By combining the predetermined resource capabilities of each grid node \( C_m \), the the overall computational capability of the grid environment is captured as:

\[
r = \{ r_{C_1}, r_{C_2}, ..., r_{C_{|C|}}\} \tag{1}
\]

In order to make service configuration decisions for the above defined computational capability of the grid environment, the performance or utility of each service \( S_i \) is evaluated by a function of quality:

\[
Q_{S_i} = Q_{S_i}(r_{C_m, S_i}) = \sum_{j=1}^{|Q_i|} w_{Q_{i,j}} * Q_{S_i,j}(r_{C_m, Q_{i,j}}) \tag{2}
\]

where \( \sum_{j=1}^{|Q_i|} w_{Q_{i,j}} = 1 \); and \( Q_{S_i,j}(r_{C_m, Q_{i,j}}) \) is a quality function that estimates the quality \( Q_{i,j} \) of the service \( S_i \) from the available resource set \( r_{C_m, Q_{i,j}} \). The quality function \( Q_{S_i,j}(r_{C_m, Q_{i,j}}) \) is further defined as:

\[
Q_{S_i,j} = Q_{S_i,j}(r_{C_m, Q_{i,j}}) = \inf\{ q_{S_{i,j}}(R_{C_m,1}, ..., Q_{i,j}, R_{C_m, Q_{i,j}}) \} \tag{3}
\]

where \( q_{S_{i,j}}(R_{C_m,k}) \) is a quality-resource function [12] which estimates the quality \( Q_{S_{i,j}} \) of the service \( S_i \) from the resource, \( R_{C_m,k} \).

To provide a clear distinction between the amount of resource \( R_{C_m,k} \) consumed by any service \( S_i \) and the total resource available, we use the terms \( R_{S_i}^{C_m,k} \) and \( R_{avail}^{max} \) respectively. That is if \( R_{S_{i,j}} \) be the amount of resource needed to deliver the quality \( Q_{S_{i,j}} \), the total amount of resource consumed by the service \( S_i \) is defined as:

\[
R_{S_{i}}^{C_m,k} = \sup\{ R_{S_{i,1}}^{C_m,k}, ..., R_{S_{i,|Q_i|}}^{C_m,k} \} \tag{4}
\]

In the above equation (4), the value of \( R_{S_{i}}^{C_m,k} \) is 0 if the service \( S_i \) is not placed on the capsule \( C_m \) or if the resource \( R_{C_m,k} \) does not provide any value to the service \( S_i \). That is, when all services are deployed in the environment, the final amount of any resource \( R_{C_m,k} \) allocated to a set of services in \( S \) is defined as:

\[
R_{S_i}^{C_m,k} = \sum_{i=1}^{|S_i|} R_{S_i}^{C_m,k} \tag{5}
\]

In order to estimate the usability of any application service configuration for the above defined computational capability of
the grid environment, we need to define an objective function under conditions that the aggregate resource consumption of the services combined in a capsule can not exceed what is physically offered by the executing node of that capsule.

A. The Optimal Service Configuration

The optimal service configuration is the one that simultaneously tries to maximise the quality levels for all services from the given set of available computational and communication resources of the grid environment $r$. By using the general practice of weighted-sum approach, the overall objective function for a set of services $S$ can be defined as:

$$\text{Maximise } \sum_{i=1}^{\vert S \vert} w_{S_i} \cdot Q_{S_i}$$  \hspace{1cm} (6)

where $\sum_{i=1}^{\vert S \vert} w_{S_i} = 1$; and the natural constraints subjected to the defined objective are:

$$R_{C_i, k}^S \leq R_{C_i, k}^{avail} \quad \forall \quad k = 1, \ldots, \vert R_m \vert \quad m = 1, \ldots, \vert C \vert$$  \hspace{1cm} (7)

$$Q_{S_{i,j}} \geq Q_{S_{i,j}}^\text{min} \quad \forall \quad j = 1, \ldots, \vert Q_i \vert \quad i = 1, \ldots, \vert S \vert$$  \hspace{1cm} (8)

In other words the optimal service configuration $\emptyset$ is a partition of the service set $S$. $\emptyset = \{\emptyset_1, \emptyset_2, \ldots, \emptyset_{\vert C \vert}\}$ such that $Q_{S_{i,j}}^\text{min} \leq Q_{S_{i,j}} \leq Q_{S_{i,j}}^\text{max}$ for all $i$ and $j$. $\emptyset_i \subseteq S$, and $\bigcup_{i=1}^{\vert C \vert} \emptyset_i = S$ and $\emptyset_i \cap \emptyset_j = \emptyset$ for all pairs $i \neq j$. Here $Q_{S_{i,j}}^\text{min}$ and $Q_{S_{i,j}}^\text{max}$ are the minimum and maximum required quality values for the service $S_i$.

B. The Feasible Service Configuration

The combinatorial explosion ($\vert C \vert^\vert S \vert$ in worst case) makes the above service configuration a complex task, and this process of application service configuration has been proven to be NP-hard [10], [9]. In the context of service partitioning problem, the objective is to seek a feasible solution instead of an optimal solution. The reason why this should be, will be explained in sections III. A feasible service configuration is the one that tries to achieve at least minimum required quality levels for all services from the given set of resources. That is, any service $S_i$ involved in the feasible service configuration should satisfy the following condition:

$$Q_{S_{i,j}} \geq Q_{S_{i,j}}^\text{min} \quad \forall \quad Q_{i,j} \in Q_i$$  \hspace{1cm} (9)

In other words, the objective of the quality-aware application service partitioning process is to determine a mapping $P : S \rightarrow C$ between the service set $S$ and the computing node set $C$ such that $P(S_i) = \{C_m \mid \exists_{C_m, S_i} \in \exists_{C_m}, \quad Q_{S_{i,j}} \geq Q_{S_{i,j}}^\text{min} \quad \forall \quad Q_{i,j} \in Q_i\}$.

C. Centralised Vs Decentralised Service Partitioning

Besides just defining the objective of the given application service partitioning problem, performance of any given solution approach depends on the algorithm used and also on the design of different decision making elements. Depending on how these elements are engaged in service configuration process, we distinguish the decentralised service partitioning from the centralised one.

In centralised service partitioning, one coordinating agent is responsible for making global decision. Though there may exist several other elements (or agents), the coordinating agent running in a capsule has a global control that centrally decides service placement in different capsules. For this, the coordinating agent is informed of different service placement alternatives in different contexts including other capsules’ resources. Despite its simplicity, centralised partitioning approaches have well known drawbacks. For example the extensive communication needs between the coordinating agent and other capsules makes the planning process non-scalable. Furthermore, centralised planning approach is intolerant to a failure of the coordinating agent.

Unlike the centralised approach, the elements in the decentralised scheme are more independent and the decision process is distributed and localised to several autonomous agents. In other words, an agent is responsible for making at least one service placement decision. The convergence of the overall system is seen as a combined behaviour of all agents involved in the service partitioning process. For this, agents communicate with other through a coordination technique. The method each autonomous agent uses to coordinate with other agents may depend on the algorithm used.

III. Complexity

The application partitioning problem defined in the section II is a multi-constraint multi-objective optimisation problem. In its primitive form it can be seen as a knapsack problem for each capsule, however with the constraint that the inclusion of one service in the subset hosted by a given capsule removes this service from all other capsules. Consequently, the knapsack problems can not be solved independently, and to our knowledge there are no good algorithms available to solve the partitioning problem in polynomial time.

Consider a problem consisting of $\vert C \vert$ capsules labelled $1, \ldots, \vert C \vert$ and $\vert S \vert$ services. A configuration is a particular assignment of the services to the capsules and can be thought of as a vector with $\vert S \vert$ elements where each element of the vector takes an integer value $1, \ldots, \vert C \vert$ indicating the current assignment of the corresponding service. Since there will be $\vert C \vert^\vert S \vert$ possible assignments for each service, there will be $\vert C \vert^\vert S \vert$ different configurations. Even though some of these configurations may be illegal, like placing a memory demanding service on a capsule with little memory, an exhaustive search of all these configurations is only possible for few capsules hosting an application with a small number of services. Even enumerating them takes too long.
Fortunately, one is not interested in the best configuration since any feasible solution will do. There may be many feasible configurations interspersed among the possible configurations, and multiple heuristics have been proposed to search the configuration space. Fundamentally, one moves from one configuration to the next by changing one element of the configuration vector, that is moving one service from its present capsule to an alternative capsule. The different algorithms can be classified according to how much and how well they exploit the information available in the capsule graph and the application graph in making the simultaneous selections of which service to move and where to move it.

A completely random algorithm will select a random service and move it to a random capsule, i.e. it moves randomly from one configuration to another. Unfortunately, it will also be memory less so seen configurations may be repeated. Hence, the number of configuration changes necessary to observe all possible configurations is a stochastic variable. The mean number of configuration changes necessary is [1]

$$E\{No.\text{moves}\} = N \log N [\exp C]^{|S|}$$

so in general this random search is $|S| \log |C|$ times worse than an exhaustive search.

The problem of service partitioning outlined above is similar to the famous mapping problem from distributed computing and the scheduling problem. Many different heuristics have been applied to these classes of problems and they can probably all easily be modified to work in this setting, although we are not aware of any such adaptation. There is an interesting comparison of algorithms for scheduling in heterogeneous grid environments [2] concluding that genetic algorithms might be slightly better than simulated annealing. The present authors have been looking at alternative approaches. [12] uses a fixed structure learning automata approach to solve the problem, while an agent based approach inspired by randomly exploring ants was used in [11]. Some of these ideas will be described in section V. If it could be possible to limit the search region, more efficient searches may be envisioned. Unfortunately there seems to be no way to characterise a feasible configuration without evaluating it against the available resources offered by the capsules. In the same way it is hard to say how many feasible solutions that may recede within the $|S|$-dimensional space. Hence, algorithms from mathematical optimisation like interior point algorithms [14, part 1] and simplex algorithms [14, part 3] might not work well in combinatorial problems like the present one, even though they can be combined with branch and bound type algorithms to provide integer solutions.

IV. APPLICATION SERVICE RECONFIGURATION AND ADAPTATION

Apart from the initial effort to partition a distributed application, any possible changes in underlying available resources of the grid environment may disturb the configured application making it perform unacceptably or even malfunction. To repair the disturbed configuration, we need to employ a reconfiguration and adaptation technique, which fundamentally corresponds to a process of finding a new feasible configuration or even modifying the existing one based on fluctuations in the system resource availability. In other words, distributed applications need to be reconfigurable to sustain minimum quality requirements and adaptive to provide maximum user satisfaction.

Within a reconfigurable and adaptive system, each distributed application will generally be subject to a control loop in which any degradation of the application performance and changes in the resource availability are detected by a surveillance method. These changes are analysed to determine whether some corrective actions on the application are needed, and if so, a set of corrective actions is determined and performed on the application. For example, the failure of a service may cause failure or degraded performance of the entire application. Therefore, provisions must be made to support corrective actions to recover from failure. Though the task of such corrective actions is specific to a given application, selection of these actions can be made using a set of situation-action rules or a utility function.

Fig. 1. A block diagram showing the connection between application reconfiguration and adaptation

Reconfiguration is the corrective action of choice in response to a service failure in the running application. The cause of the failure might be a sudden brake-down of a capsule or a shared-link between any two capsules or even a drastic drop in resource availability. Service failures due to low resource availability can be addressed by adjusting or transferring a small portions of resources used by other services that are running on the same capsule or sharing the same communication links. However if this is not possible because of insufficient amount of resources, the necessary corrective action is principally the same as for the case of capsule or communication link failures. Recovery from capsule or link failures is typically provided by replication of the same service to several capsules using techniques such as active and passive replication. When a capsule hosting a replicated service fails, it becomes necessary to restart the corresponding failed service replicas in other capsules. The state of the restarted service replicas may be recovered from the service replicas that did not fail. Ideally the same algorithm used for finding the initial placement of the services may also be reused in this case to find a new location for failed services. The only difference is that the input to the algorithm will be a service configuration in which all services except the failing ones, already have
a placement satisfying their minimum quality requirements. It therefore seems probable that the effort of finding a new feasible configuration on average will be much less than the effort of finding an initial feasible configuration. However in some specific cases where any replacement of failed services is not possible, may cause other services to be replaced and this may further lead to a deadlock situation. In such complicated cases, the entire application system should be rebuilt, and this is equivalent to the initial service partitioning process. This may perhaps be impractical for large applications that require a large amount of time to configure.

When applications are seen as a composition of individual service components, it is also possible that there may exist multiple service compositions that can satisfy the required application functionality. In such cases, the service reconfiguration may also include addition, removal or replacement of services that will affect structure of the service composition in order to sustain the minimum quality levels of a running application. However our focus in this paper is limited to a single service composition and the method of reconfiguration discussed here ensures that the set of service components involved in the application is fixed and the reconfiguration is strictly a healing process from a failure in a running application [8].

Adaptation is the corrective action in response to any increase or decrease in resource availability. This happens when a running application service realises that there is an increase or decrease in the current resource availability. The reason for this may be that the other applications running in the shared grid environment terminate and the resources assigned to them are released. In this case the application services running in different capsules, should have the potential to improve their performance by increasing their quality levels. Otherwise, in the case of decrease in resource levels due to the start of new applications, the service should be restricted to use at least minimum required resources.

Our definition of adaptation clearly differs from the process of reconfiguration, since adaptation aims to improve the overall performance of the application without making any changes to the current placement of services while reconfiguration is more concerned about sustaining minimum service quality levels and it takes place only if any failures or chances of a failure are detected. Our distinction of adaptation from reconfiguration is made based on the following two reasons:

- Deviation between the quality levels of any two services should be minimised (tuning).
- Quality level of each service should be maximised.

Any feasible service configuration generated during application configuration and reconfiguration process may not guarantee the overall performance of the application. This is because of high variations between the quality levels of services in the service set, S. For example in distributed real-time applications like video streaming service [5], when some data is sent continuously from one service to another, even a small quality difference between the services may cause input queues build up on the receiving side. Such differences between the services even with acceptable minimum quality may degrade the overall performance of the application. For this we need an adaptation technique that will minimise differences between the quality levels of all services participating the application. In other words, quality levels of all services in S must be synchronised. That is:

\[
\text{Minimise } |Q_j - Q_i| \quad \forall \ i,j \in [1,|S|] \quad (10)
\]

Unlike reconfiguration, the objective of adaptation process is to analyse the current running configuration and exploit the opportunities to make the running application more efficient in performance by making best use of available resources. When an application is ready to be deployed on a distributed computing platform, it is highly desirable to see a feasible configuration, since finding an optimal configuration might take infinitely long and also there is no guarantee that the resource capability r of the grid environment will not change. But once the application is configured or reconfigured and running in a grid environment, the adaptation process can take place in the background and propose necessary changes in resource allocation so that the overall performance of the application will be improved.

To put it in a formal way under the assumption that capsules are the containers for hosting the services and each capsule can access the communication links from other, the combined quality of each capsule \(C_m\) as a function of qualities of the individual services on that capsule, should be maximised. That is:

\[
\text{Maximise } \frac{1}{|S_m|} \sum_{i:S_i \in S_m} Q_i \quad \forall \ m \in [1,|C|] \quad (11)
\]

V. NEW BRANCH OF EVOLUTIONARY ALGORITHMS

Apart from the discussion presented in section III, we focused on a new branch of evolutionary algorithms that fall into the category of decentralised solution techniques. The strong motivation behind these decentralised algorithms is the success
of autonomous agents in solving multi-objective optimisation problems [3]. In autonomous agent-based computing, each agent interacts with some environment, makes an action (local decision) in order to meet its own objectives while interacting with other agents in the environment such that the global objective of the system is achieved. Any decision made by an agent depends on its local environment and the interests of other agents. That is, each agent needs to interact with other whose local decisions are interdependent on the agent’s local objective reflecting its own individual interests. Such an interaction may provide some partial information of other’s interests and resources within non-local environment. This allows the agent to choose the next most appropriate action. In [7], it is also emphasised that the coordination between the agents, prevent individuals from working in isolation and ending up in different conflicting groups that will not converge to a global solution. Therefore the use of such coordination is a most influential feature of the decentralised problem solving process. In this section we present two advanced learning and nature inspired solution techniques that are able to find a feasible configuration by achieving the minimum quality objectives of the composed application service and capable of adapting the dynamically changing grid environment with minimum changes. These algorithms have been selected because they are capable of working with distributed decisions in random environments.

A. Learning Automaton-Based Solution

Having motivated by the fact that the fastest known equipartitioning algorithm is based on a fixed structure learning automata [13], we adapted a learning automaton solution technique to solve the service partitioning problem. Basically a learning automaton is a software agent which is assigned with some task to complete in a computational environment. A learning automata basically proposes an action to a stochastic environment that returns a binary feedback (reward or penalty). Through several such iterations the automaton will learn which action is most likely to give a positive feedback. The theory of learning automata have also been successfully used to map the processes of a parallel application onto processing nodes [6], and this problem resembles the one at hand. Within our quality-aware service partitioning setting, automata are represented by services and the environment by a set of capsules. The action is equivalent to choosing a capsule of the grid environment. The environment’s feedback is produced based on the output from quality objective functions defined in the section II.

Using this approach, individual services in the service set interact with the grid environment to make service configuration decisions. Such a service configuration is done in an autonomous fashion to improve the scalability of distributed service planning. The proposed learning automaton based solution [12] establishes an autonomous learning environment for the services. That is, as long as the current mapping or partition is not feasible, the unsatisfied services of the current partition move between the capsules. During this process all services learn and even unlearn from behaviour of the grid computational environment. Such a learning process of each service is done by making changes in its confidence level. That is, any positive feedback (reward) from the grid environment allows a service $S_i$ to increase its confidence, similarly a penalty forces the service to decrease its confidence. To make this happen, the learning algorithm follows a specific strategy. More detailed description of the proposed learning automaton based solution can be found in [12].

B. Ant-Based Solution

![Fig. 3. An example of a service partition. The small dots are ants and the circles are the reported locations of the services moving to new capsules when the concentration of its ants in that capsule exceeds a certain threshold.]

The ant system (AS) constitutes of a set of autonomous agents that imitate the behaviour of real ants of an ant colony [4]. Given the objective of an optimisation problem, the general functionality of the ant system (AS) can be described in terms of a set of multiple autonomous agents (ants) that leave the nest with an objective to find a feasible point (food) in the solution space. Each ant moves among capsules following the edges of the graph, and this will continue until the given task is completed. Within our quality-aware application service partitioning context, each service employs several ants that work for their employer service. Further the environment is represented by a set of capsules. The objective of any ant is to visit and check the possible placement of its owner service with different capsules of the grid environment. As a result each ant carries the environment’s feedback to its parent service. In other words, multiple ants that are set to explore different parts of the solution space while cooperating with each other. The proposed ant-based algorithm in [11] establishes a de-centralised solution technique to partition the services. The algorithm proposes to create multiple ants for each service and these ants forage to find the places (capsules) where the quality of their service is satisfied. Each ant has a life span where it can at least make a certain number of moves. That is, as long as the current mapping or partition is not feasible, the ants move between the capsules.
During this process, a service move to the capsules where the concentration of its ants is high. The process is illustrated in Figure 3. After the move of a service, its new position (capsule) is broadcast to all ants in the system for them to take this changed position into account when computing the quality level of their services. Finally the global state of the application and the feasibility search process is driven by an asynchronous updating scheme which organises individual ants.

VI. CONCLUSION

In this paper we have addressed the problem of allocating a set of connected services onto a grid environment of networked capsules, and subsequently re-partition or adapt the service set as a consequence of dynamic changes in the underlying resources of the grid environment. Due to the complexity of the problem at hand, we focused on the problem of finding a feasible partition and not necessarily an optimal one. We discussed a general case of decentralised agent based solution approach that uses autonomous agents in solving multi-objective optimisation problems. This has motivated us to search for a new branch of evolutionary algorithms. We have therefore proposed two special type of evolutionary algorithms that seem promising for both handling the complexity of the service partitioning and the dynamic behaviour of the grid environment.

REFERENCES