Applying Particle Swarm Optimization to Quality-of-Service-driven Web Service Composition

Simone A. Ludwig
Department of Computer Science
North Dakota State University
Fargo, ND, USA
simone.ludwig@ndsu.edu

***Abstract*—Web service composition is a very important task in service-oriented environments. The composition of services has to be based not only on functional but also on non-functional properties. In particular, the selection of the services should be performed at run-time rather than at design-time in order to adjust to changes in the environment that are due to the volatile nature of service-oriented environments. An optimization technique is necessary to perform the composition of web services based on Quality of service (QoS) parameters. Many different methods have been used to address the service composition problem; in particular, many linear programming methods have been used to optimize the process of service composition. The proposed work is different from related work in two aspects: (1) two meta-heuristic methods based on Particle Swarm Optimization (PSO) are introduced to address the optimization problem, and (2) several workflow requests are being processed simultaneously. The experimental results show that the hybrid PSO method in particular performs very well in service-oriented environments.**

particle swarm optimization; quality of service; workflow composition

I. INTRODUCTION

Services computing is a discipline combining science and technology. In particular, it is a bridge for the gap between Business Services and IT Services. The core technology suite includes Web services and service-oriented architecture (SOA), cloud computing, business consulting methodology and utilities, business process modeling, transformation and integration [1]. Services computing covers the whole life-cycle of services innovation research. It includes business componentization, services modeling, creation, realization, annotation, deployment, discovery, composition, delivery, service-to-service collaboration, monitoring, optimization, as well as management. The major goal of services computing is to enable services and computing technology to perform more efficiently and effectively [1].

Services are components that support the composition of distributed applications, and are offered by service providers/organizations that procure the service implementations, supply their service descriptions, and provide related technical and business support. Since services are offered by different businesses and communicate via the Internet, they provide a distributed computing infrastructure for both intra- and cross-enterprise application integration and collaboration. Descriptions of services are used to advertise the capabilities, interface, behavior, and quality of the service. Service descriptions are necessary for discovery, selection, binding, and composition of services. The service capability description presents the conceptual purpose and expected results of the service. The service interface description publishes the service's input, output, and message types. The service behavior description describes the "expected" behavior of a service during its execution. Finally, the Quality of Service (QoS) description publishes important functional and non-functional service quality attributes, such as cost, performance metrics, security attributes, reliability, scalability, and availability. Service clients, and service workflow aggregators utilize service descriptions to achieve their objectives [2].

Authors in [3] have identified the following components within service-oriented computing:

Service foundations: consists of service-oriented middleware, which provides the runtime SOA infrastructure connecting heterogeneous components and systems and providing access to services over various networks such as the Internet. The middleware allows application developers to define basic service functionality describing, publishing, searching and binding of services. The overall use of the service-oriented architecture is for services to describe their capabilities in a registry, so that a user can search and discover the appropriate service based on the capabilities.

Service design and development: The design of services is a very important component of a service-oriented architecture. A well-designed infrastructure can increase the efficiency of businesses by providing reusable, independent, automated processes as services and mechanisms to effectively use them.

Service management and monitoring: Service management is responsible for the installation and configuration of metrics such as QoS during service execution, in order for the system to be able to monitor the events of the services. Service monitoring includes the collection of information produced by the services and business process, viewing the statistics of process instances including the number of instances in each of the four states (running, suspended, aborted or completed), in order to intervene in the different states of the process instances.

Service composition: Most of the time a single service will not provide the functionality needed. Therefore, the composition of several services is necessary to produce the needed functionality. In addition, in some domains complete process flows, also referred to as workflows, are needed to provide the envisioned capability.

The composition process can be performed either manually or automatically, i.e. the composition can be done during the design time, which is also referred to as static composition, or it can be done at runtime using a dynamic composition process. This composition process involves the creation of a composition schema in order to satisfy a series of goals set by a user, which is a multifaceted problem, since many different issues need to be dealt with at once. First of all, an ever-growing service repository needs to be searched in order to find the appropriate services that satisfy the user's requirements. Also, assuming that the services have been found, the successful combination, any resolution of potential conflicts and inconsistencies have to be addressed given that most of the services have been created by different vendors using different implementation methods and systems. Furthermore, since inconsistencies may occur at runtime, it might become necessary to predict such events in order for the system to react and not fall apart during such encounters. In addition, even after these issues have been addressed, the system should be able to adapt to the dynamic characteristics of service-oriented systems, namely services going online or offline, new services becoming available, or existing services changing their characteristics.

This paper addresses the service composition task introducing Particle Swarm Optimization (PSO) to achieve the selection. Given that this optimization problem was mainly tackled with linear programming methods, as well as some Genetic algorithm approaches, PSO has been investigated in this paper. Most past literature only considered one workflow request at a time, whereas our approach serves several requests simultaneously. We make a case for using a meta-heuristic method to this assignment problem, considering also the scalability of the approach.

The remainder of this paper is structured as follows: Section II reviews related work, in Section III the web service composition model is outlined and an example workflow is presented, as well as the discrete guaranteed convergence particle swarm optimization approach, and the hybrid particle swarm optimization approach are presented. In Section IV an empirical study is conducted and results are outlined and discussed, and Section V concludes this paper with a critical analysis of the approaches applied to service composition.

II. Related Work

Before focusing on different composition approaches a description of the different composition models is provided, which are orchestration, choreography, coordination and component. Orchestration [4] is a description of how services that participate in a composition interact at the message level, including the order in which iterations should be executed as well as the business logic. Therefore, orchestration defines the interaction with the services it orchestrates, before any actual execution takes place. One of the languages for defining choreographies is the Web Service Choreography Description Language (WS-CDL) [5].

Service coordination groups a set of service instances together based on a coordination protocol. This protocol describes the way the participating services should interact, what the outcome of the interaction should be, and whether it was successful or not. The coordinator, which is a third party, acts as the nexus between participants and is responsible for the correct flow of the encounters, as well as the decision and dissemination of the outcome once the activity ends. OASIS has developed the Web Services Composite Application Framework (WS-CAF [6], which includes the Web Service Coordination Framework (WS-CF).

The component model, also referred to as service wiring, is concerned with the actual linkage of the services during the composition. Service wiring is described in the Service Component Architecture (SCA) [7], in which the service components can be implemented in any programming language, and are encapsulated in order for the components to be able to share a similar description.

Automated web service composition approaches can be grouped into workflow-based, model-based, theoretic-based and planning-based approaches. An example of a workflow-based approach is a framework that automatically constructs a Web service composition schema from a high-level perspective [8]. The input is fed to an abstract workflow generator that attempts to create an abstract workflow that satisfies the objective by either using already generated workflows or subsets of them that are stored in a repository or by performing backtracking to find a chain of services that satisfy the objective. The identified abstract workflow is then instantiated by either finding existing services through a matchmaking algorithm that matches inputs and outputs and binds them to the workflow, or by recursively calling the abstract workflow generator if no service can be found for an activity.

Model-based approaches are manifold. An e-service composition tool [9] implements an automated service composition using Finite State Machines (FSMs). The behavior of a service is modeled using two schemata; an external schema specifies its exported behavior and an internal schema contains information on which a service instance executes each given action that is part of the overall service process. When the composition is synthesized, the external FSM models of the available services, and target services are transformed to modal logic formulas in Deterministic Propositional Dynamic Logic (DPDL). If the resulting set of equations is satisfiable, then a FSM for the target service exists. The synthesized FSM can then be converted to a BPEL process and executed in an engine. Some recent work of the authors, replace the FSMs with transition systems and use a simulation to virtually compute all possible composition at once.

Theoretical-based approaches use process algebraic languages, such as CCS (Calculus of Communicating

Systems) [10] or pi-calculus [11] since the process specifications included in Semantic Web frameworks, such as WSMO and SWSF, are formally grounded on process algebras. Pi-calculus as described in [12], can be used to describe, as well as compose Web services, and the processes can be sequences of other processes, parallel compositions, recursive executions, or choices between operations, and is therefore able to express all basic composition schemas. Information exchange is done between processes as input and output artifacts, which are exchanged on channels.

Planning is introduced in traditional workflow-based services composition [13]. Planning techniques to concretize the workflow are used given an abstract BPEL flow. OWL-S service descriptions are translated to the Planning domain and problem description languages. PPDL descriptions are then fed to IBM's Planner4J planning framework containing many planning algorithms, including the classical planning ones. At runtime, a concrete service is requested from the planner module given each task in the abstract BPEL flow. If a service is not found, the planner attempts to solve a planning problem with the available services as input. However, this approach is only semi-automatic since the creation of the abstract flows is manual.

Another classical planning approach is proposed in [14]. A formulation of service composition as a goal-directed planning problem is described, which takes three inputs that are a set of domain specific service composition rules, a description of a business objective or goal, and a description of business assumptions. An ontology represents these concepts and a three-step composition schema generation is conducted. The composition rules are exploited to create a chain starting from the business objective and moving backwards until there are no more rules or the initial state is reached as the first step, which is also referred to as Backward-Chaining. The second step, called Forward-chaining, follows the opposite direction and attempts to complete the composition schema produced by adding services that may be required by the results of some tasks. The final step is the data-flow-inference phases, which adds a data flow to the composition schema.

Related work in the area of composition of workflows with particular focus on Quality-of-Service (QoS) is multifaceted. In [15], complex workflow patterns are used to address the service selection problem. Linear programming is used to solve the optimization problem using an aggregation function for different QoS attributes.

A framework for QoS-based web service contracting is proposed in [16]. The framework consists of an extensible model that defines domain-dependent and domain-independent QoS attributes, as well as a method to establish the contract phase is proposed. A matchmaking algorithm ranks the services that are functionally equivalent based on their ability to fulfill the requirements.

A dynamic web service selection and composition approach is described in [17]. The approach determines a subset of web services to be invoked during runtime in order to orchestrate a composite web service successfully. A finite state machine model is used to describe the permitted invocation sequences of the web service operations, and a reliability measure is aggregated for the web service operations.

In [18], a quality-driven web service composition approach is outlined applying linear programming. A global planning approach is employed to optimally select component services during the execution of a composite service.

Given that linear programming approaches do not scale very well with increasing problem sizes (increasing numbers of workflows and services), approximate techniques are necessary. In this work, we propose the use of PSO and implement one basic PSO and one hybrid PSO in order to compare their performance on the workflow composition problem. Furthermore, we process and optimize several workflow requests simultaneously.

III. Web Service Composition Model

We have devised a web service composition model as outlined in Figure 1. As we are primarily interested in the optimization of the incoming user requests, we make the following assumptions. The workflow database is in place and provides several options of the service flow if these exist. Also, we are assuming that the users' requests are being simultaneously processed in certain time intervals, i.e. we are optimizing several workflow requests at the same time. In addition, we assume that more services are available than requests, i.e. every request can be satisfied in terms of available services.

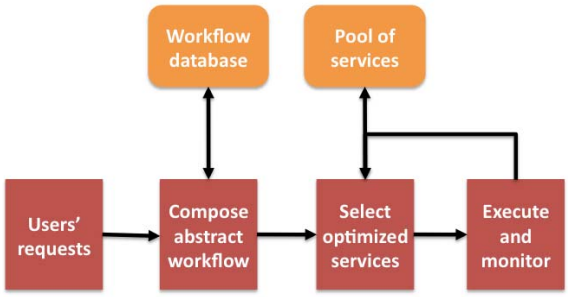

Figure 1. Web Service Composition Model.

The figure shows that the users' requests are entered and the system starts composing abstract workflows using the workflow DB. The workflow DB contains all possible abstract workflows for the specific requests, i.e., compositions based on functional requirements are provided. Given the preferences on QoS criteria by the users, the concrete service instances for the different workflows are optimized. An optimization method is run to satisfy the users requests together with the workflow instantiations, in order to return the users' workflows, which satisfy their needs. These instantiations of the services are then executed, and are monitored for the event an error or failure occurs. The type of failures that can occur during service execution are services not being available anymore, services not providing the promised QoS any longer, services not providing the functionality anymore, etc. For

these events there needs to be a fall back method in order to use a different instantiation of services within the workflow. In the following, we only focus on the optimization component.

A. *Workflow Example*

In order to show how abstract workflows are provided with the help of the workflow DB and the user's input request, the following sample workflow as shown in Figure 2 is provided, displaying the abstract as well as the concrete services.

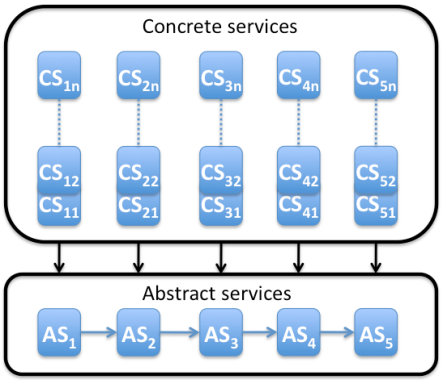

Figure 2. Example of an abstract image processing workflow.

The figure displays an example of the processing of an image [19]. The process is as follows: the image is first read in (AS_1), and converted to Grey scale (AS_2), then it is thresholded (AS_3), i.e., image pixels less than a certain amount is set to black and everything higher is set to white. The difference is then taken between this image and an image where the white lines are thinned out, i.e., detail is taken away (AS_4). The resulting image is produced, by taking the difference between the detail of the white parts, which were pruned by the *ShrinkWhite* unit resulting in an outline of an image, and saved as a new compound component (AS_5).

The concrete services (CS) for each abstract service provide the same functional, but different non-functional properties; i.e., QoS attributes. Selecting the services of a workflow based on QoS parameters requires an algorithm that can optimize the assignment of concrete services for a workflow for a given abstract workflow description. Furthermore, we are considering that multiple requests are being served at the same time. Given that performance in a service-oriented environment is of essence, we argue that we do not need to have optimal assignments, but close to optimal assignments should be found within a reasonable time.

B. *Quality of Service Metric and Objective Function*

There are many measures available for different QoS criteria, however, we consider the following five generic quality criteria for single services, also referred to as QoS parameters: reliability, availability, reputation, execution duration, and execution price.

The *reliability* $q_1(s)$ of a service is the fraction of requests correctly responded to within a maximum expected time frame. Reliability is a measure related to the hardware and software configuration of Web services and their network connections. Reliability values are computed from past data measuring the successful executions in relation to the overall number of executions.

The *availability* $q_2(s)$ of a service is the fraction of time that the service is accessible. It basically measures the fraction of the total amount of time in which the service is available during the last defined period of time (threshold is set by administrator).

The *reputation* $q_3(s)$ of a service is a measure of its trustworthiness and depends on the end user's experience of using a service. The value of reputation is defined as the average ranking given to a service by different end users.

The *execution duration* $q_4(s)$ denotes the expected delay in seconds from the moment a request is made until the moment when the results are returned. Services advertise their processing time or provide methods to inquire about it.

The *execution price* $q_5(s)$ represents the amount of money a user has to pay for executing a service. Web service providers usually advertise the execution price directly or they provide methods to inquire about it.

The QoS vector $q(s)$ of a service s is defined as follows:

$$q(s)=(q_1(s),q_2(s),q_3(s),q_4(s),q_5(s))$$

However, in this study we are concerned not only with single services but with complete workflows, and therefore the QoS parameters of the single services have to be aggregated. We assume in our study that we are only using sequential workflows. Therefore, the *availability* $Q_1(wf)$ and *reliability* $Q_2(wf)$ of a workflow wf is calculated as the product of each single service's availability and reliability respectively. The *reputation* $Q_3(wf)$, *execution duration* $Q_4(wf)$, and *execution price* $Q_5(wf)$ of a workflow wf is the average of each single service's reputation, execution duration and service cost respectively.

Therefore, the QoS vector $Q(wf)$ of a workflow wf is denoted as:

$$Q(wf)=(Q_1(wf),Q_2(wf),Q_3(wf),Q_4(wf),Q_5(wf))$$

Our goal is to maximize the overall QoS not only for one workflow, but for N workflows simultaneously. The overall objective function for the optimization of the workflows is the following:

$$f_{obj}=\max \sum_{i=1}^{N}\left(\sum_{j=1}^{3} w_j \frac{Q_{ij}-Q_j^{\min}}{Q_j^{\max}-Q_j^{\min}}+\sum_{j=4}^{5} w_j \frac{Q_j^{\max}-Q_{ij}}{Q_j^{\max}-Q_j^{\min}}\right)$$

whereby we define Q_{ij} to be the value for workflow i and the j^{th} QoS parameter, and we define $Q_j^{\max}$ to be the maximum score any of the considered services achieves for the j^{th} QoS parameter as defined above, i.e., $Q_j^{\max}=\max_{s\in S} q_j(s)$ where S is the set of all possible services. And similarly, $Q_j^{\min}$ to be the minimum score any

of the considered services for the j^{th} QoS parameter ($Q_j^{\min} = \min_{s \in S} q_j(s)$).

Note that the individual QoS parameters are treated differently depending whether its value is to be minimized or maximized. Normalized scores are used and each QoS parameter can be weighted differently by parameter w_j.

C. Particle Swarm Optimization Approach (PSO)

PSO, as introduced in [20], is a swarm based global optimization algorithm. It models the behavior of bird swarms searching for an optimal food source. The movement of a single particle is influenced by its last movement, its knowledge, and the swarm's knowledge. In terms of a bird swarm this means, a bird's next movement is influenced by its current movement, the best food source it ever visited, and the best food source any bird in the swarm has ever visited.

PSO's basic equations are:

$$x_i(t+1) = x_i(t) + v_{ij}(t+1)$$

$$v_{ij}(t+1) = w(t)v_{ij}(t) + c_1 r_{1j}(t)\left(xBest_{ij}(t) - x_{ij}(t)\right) + c_2 r_{2j}(t)\left(xGBest_j(t) - x_{ij}(t)\right)$$

where x represents a particle, i denotes the particle's number, j the dimension, t a point in time, and v is the particle's velocity. *xBest* is the best location the particle ever visited (the particle's knowledge), and *xGBest* is the best location any particle in the swarm ever visited (the swarm's knowledge). w is the inertia weight and used to weigh the last velocity, c_1 is a variable to weigh the particle's knowledge, and c_2 is a variable to weigh the swarm's knowledge. r_1 and r_2 are uniformly distributed random numbers between zero and one.

PSO is commonly used on real and not discrete problems. In order to solve the discrete assignment problem using the PSO approach, several operations and entities have to be defined. This implementation follows the implementation for solving the traveling salesman problem as described in [21]. First, a swarm of particles is required. A single particle represents a possible workflow, i.e., every particle's position in the search space must correspond to a possible workflow. The workflow, that is the position, is implemented as a vector. Dimensions in the vector correspond to workflows, and values correspond to single services.

Velocities are implemented as lists of changes that can be applied to a particle (its vector) and will move the particle to a new position (a new workflow). Further, minus between two particles, multiplication of a velocity with a real number, and the addition of velocities have to be defined. Minus is implemented as a function of particles. This function returns the velocity containing all changes that have to be applied to move from one particle to another in the search space. Multiplication randomly deletes single changes from the velocity vector, if the multiplied real number is smaller than one. If the real number is one, no changes are applied. For a real number larger than one, random changes are added to the velocity vector. When a velocity is added to another velocity, the two lists containing the changes will be concatenated.

The PSO implemented uses guaranteed convergence, which means that the best particle is guaranteed to search within a certain radius, implying that the global best particle will not get trapped in a local optima. Configurable parameters in the implementation include numbers of particles (size of the swarm), number of iterations, c_1 (the weighting of the local knowledge), c_2 (the weighting of the global knowledge), w (the weighting of the last velocity), radius (defines the radius in which the global best particles searches randomly), global best particle swarm optimization (determines whether global best particle swarm or local best particle swarm optimization is used), and neighborhood size (defines the neighborhood size for local best particle swarm optimization).

D. Hybrid Particle Swarm Optimization Algorithm (Hybrid-PSO)

The challenge PSO as well as other meta-heuristic algorithms face is the balance between exploration and exploitation of the search space. Exploration describes the behavior of the algorithm when searching a broader region of the search-space, and exploitation describes a focused search as to get closer to the optimum. This trade-off between exploration and exploitation is thoroughly discussed in [22-24] outlining the convergence and parameterization of PSO. However, recent research of hybrid approaches combining PSO with a local optimizer such as hill-climbing have shown to improve the performance [25]. Other examples of successfully applied hybrid PSO approaches include a hybrid algorithm combining PSO with Back-propagation [26], another hybrid approach combines Euclidian distance based genetic algorithm and PSO [27], and furthermore, a hybrid heuristic model combines PSO with simulated annealing [28].

In this study, we combine the PSO algorithm with a combinatorial optimization algorithm that was first referred to as the Hungarian algorithm, and was developed by Harold Kuhn in 1955 [29,30]. Kuhn named it "Hungarian method" because two Hungarian mathematicians, Denes Koenig and Jeno Egervary, were the first who worked on the algorithm. In 1957, the algorithm was reviewed by James Munkres and he observed its (strongly) polynomial behavior, and therefore, the algorithm was given the name Kuhn-Munkres algorithm or Munkres assignment algorithm [31,32]. The time complexity of the original algorithm was $O(n^4)$, however, it was later modified by Edmonds and Karp (and independently by Tomizawa) to achieve a runtime complexity of $O(n^3)$ [33]. As stated before, Munkres is an optimal algorithm, however, it suffers from the cubic time complexity.

Preliminary results conducted with the Munkres and PSO algorithms resulted in a long runtime and optimal workflow assignment for the Munkres algorithm, and a much shorter runtime for the PSO algorithm however with fitness values of only around 80% of the optimal value. Therefore, the combination of both algorithms is a logical step to achieve a reasonable runtime and a very close to

optimal assignment, i.e. fitness value. The PSO algorithm was combined with the Munkres algorithm as follows. The Munkres algorithm is applied after every PSO iteration. For 10% of the number of particles, one particle is randomly chosen, and thereof 10% of the workflows are also randomly chosen to perform the Munkres algorithm on this selection. If after applying the Munkres algorithm an improvement is achieved, then that particular particle is updated with the optimized workflow and the next iteration continues.

An implementation developed by Nedas in Java, which is freely available at [34] was slightly adapted and combined with the implemented PSO code as described in Section IIIc.

IV. Empirical Study and Results

A. Experimental Setup

PSO and Hybrid-PSO as introduced in the previous section were implemented, as well as the Munkres algorithm. Given that Munkres achieves an optimal assignment, this assignment is used for both PSO algorithms as the baseline for the fitness value (optimal assignment for fitness calculation). Experiments were designed to measure the overall fitness value and the execution time of all approaches. The PSO and Hybrid-PSO were further analyzed with regard to the number of iterations and the particle size used. All measurement points shown are average results taken from 30 runs in order to guarantee an equal distribution and statistical correctness. The data sets for the user requests and composed workflows were randomly generated. We generated workflows consisting of up to 5 services. Please note that we assumed that a particular service can be used in several workflows. All fitness values shown are normalized with respect to the Munkres algorithm.

The following parameters have been chosen due to their superior performance on previous runs of the workflow selection problem, balancing between accuracy and execution time, also with regards to scalability.

For the PSO and Hybrid-PSO, the parameters were set to: number of particles = 100, number of iterations = 140, c_1 = 0.5, c_2 = 1.8, w = 0.05, radius = 5.0 and using global best PSO. The Hybrid-PSO has some specific parameters set for the local search component, which are particles selected set to 10%, and the request-workflow pair selection was set to 10%.

The experiments were conducted on an Intel Core 2 Duo (2.4GHz, 3MB L2 cache) running the Java Version 1.6.0 JDK Runtime Environment.

B. Experiments and Results

First, the fitness values of all three algorithms using 100 workflows and 500 services were measured. As expected, the Munkres algorithm achieves a fitness value of 100%, whereby the Hybrid-PSO comes quite close with 94.7%, followed by the PSO with 83.9%.

Secondly, the algorithms were tested in terms of execution time. We chose the configuration of 100 workflows from 500 services to be selected. Munkres, being an optimal algorithm, had the largest execution time, followed by the Hybrid-PSO algorithm, then the PSO algorithm. The execution time for Munkres was 210.7 seconds, for the Hybrid-PSO it was 7.0 seconds, and PSO ran for 0.8 seconds.

For the next experiments both evolutionary algorithms are compared in terms of increasing numbers of iterations, as well as increasing particle sizes.

Figure 3 plots the fitness values against the number of iterations. As can be seen, the PSO algorithm slowly but steadily increases the fitness values with every iteration, however, the Hybrid-PSO algorithm shows a large increase at the beginning and is then flattening out with increasing numbers of iterations when approaching a fitness value of 100%. At iteration 100, the fitness value of Hybrid-PSO is 99.0% and the fitness value of PSO is 87.8%. The reason for the larger increase at the beginning is due to the local search component of the Hybrid-PSO algorithm.

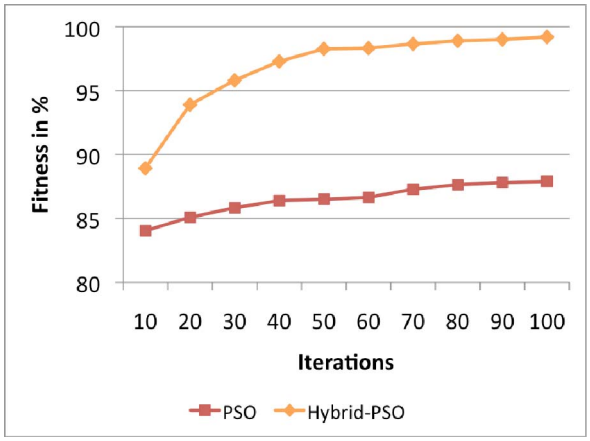

Figure 3. Fitness value vs. number of iterations of the PSO and Hybrid-PSO algorithms

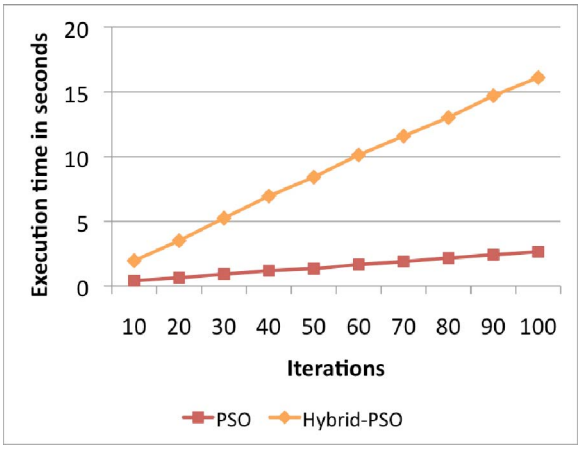

Figure 4. Execution time vs. number of iterations of the PSO and Hybrid-PSO algorithms

Figure 4 shows a linear increase in time for increasing numbers of iterations for the PSO and Hybrid-PSO algorithms, whereby the Hybrid-PSO algorithm shows the

larger increase, due to the additional time needed for the local search.

Figures 5 and 6 show the fitness value and execution time for both evolutionary algorithms when increasing the population size. In Figure 5, a slight increase in fitness value for both algorithms is shown, whereby Hybrid-PSO achieves a fitness value of 91.4%, and PSO scoring 84.7% for a particle size of 200.

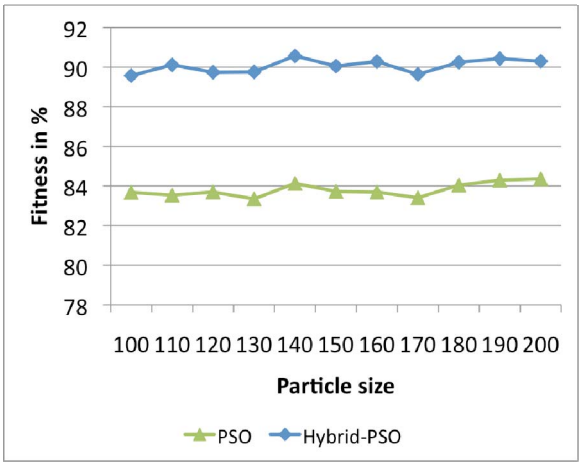

Figure 5. Fitness value vs. population size of the PSO and Hybrid-PSO algorithms

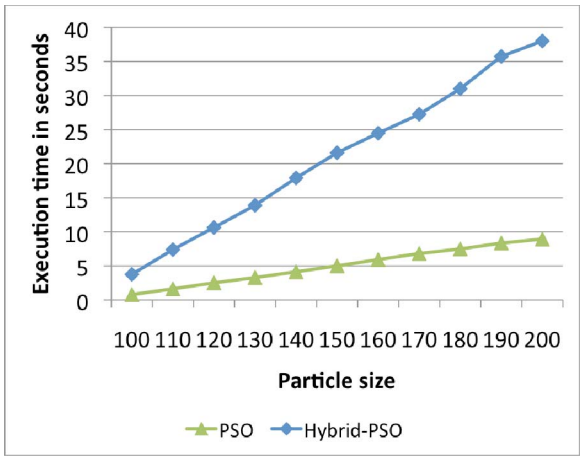

Figure 6. Execution time vs. population size of the PSO and Hybrid-PSO algorithms

Figure 6 shows, as expected, a linear increase in time for increasing population sizes, showing a larger increase for the Hybrid-PSO algorithm.

Since this investigation is primarily concerned with the scalability of the algorithms, the fitness value and execution time with increasing numbers of request-workflow pairs is explored, displaying the results in Figures 7 and 8. The request-workflow pairs were increased up to 600 services in steps of 50, and the workflows in steps of 10.

Figure 7 displays the fitness value for increasing numbers of request-workflow pairs showing relatively stable fitness values for all algorithms, however, indicating a slight decrease with larger request-workflow pairs.

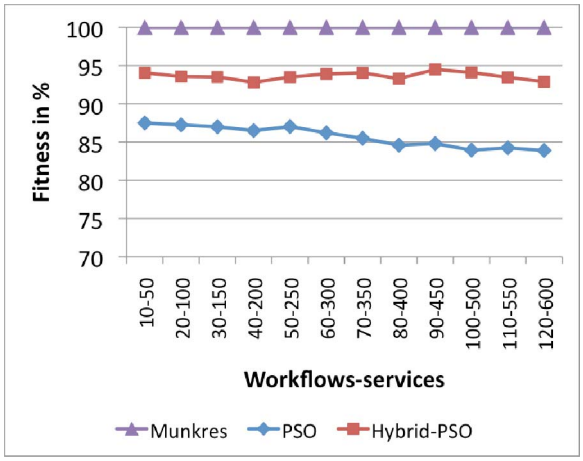

Figure 7. Scalability of algorithms in terms of fitness value

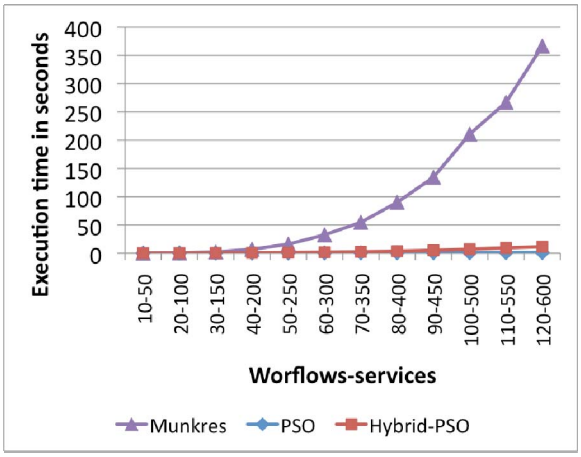

Figure 8. Scalability of algorithms in terms of execution time

Figure 8 shows the execution time of all algorithms, highlighting the Munkres' cubic time-complexity, whereas both PSO algorithms perform much faster.

V. CONCLUSION

This paper addressed the service composition task using PSO to achieve the assignment selection. These types of optimization problems were mainly addressed in the past by linear programming methods, as well as some Genetic algorithm approaches. All the past approaches only optimized one workflow request at a time, whereas our approach serves several requests simultaneously. We argue that in service-oriented architectures, several user requests need to be satisfied at any given time. In addition, given that scalability is a major issue in service-oriented environments, an approximate method should be employed for this assignment problem. Therefore, a PSO approach was investigated for its suitability. Given that PSO suffers from premature convergence, and taking the benefits from an exact method into account, we also implemented a Hybrid-PSO algorithm combining PSO with the Munkres approach.

As shown by the experimental results, the Hybrid-PSO shows a good balance between fitness value and execution time by achieving a fitness value of 94.7%; the PSO scores

83.9% for 100 workflows and 500 services. In terms of execution time and scalability, the Hybrid-PSO algorithm (even though taking extra time to perform the local search) when compared to PSO, the Hybrid-PSO still shows a linear behavior and is not much worse than the PSO (for 120 workflows and 600 services: Hybrid-PSO: 11.4 seconds, and PSO: 1.1 seconds).

Future work will follow three directions. First of all, since the workflow and concrete services were fairly similar in terms of range, the effect of larger variations in the request-workflow pairs will be investigated. Secondly, a multi-optimization PSO will be investigated and compared to the current approaches. Thirdly, different selection matching functions will be experimented with.

ACKNOWLEDGMENT

This paper is based on research supported by North Dakota EPSCoR and National Science Foundation Grant EPS-0814442.

REFERENCES

[1] IEEE Computer Society, Overview, http://tab.computer.org/tcsc/scope.htm.

[2] M.P. Papazoglou, D. Georgakopoulos, Service-Oriented Computing. Communications of the ACM, 46(10):25–65, 2003.

[3] M.P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, Service-Oriented Computing: State of the Art and Research Challenges, Computer, pp. 38-45, November, 2007.

[4] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. IEEE Computer, 40(11):38-45, 2007.

[5] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web services choreography description language version 1.0. World Wide Web Consortium, Candidate Recommendation CR-ws-cdl-10-20051109, November 2005.

[6] Organization for the Advancement of Structured Information Standards (OASIS). Web Services Composite Application Framework. Internet (http://www.oasisopen.org/committees/), 2006.

[7] O. S. Collaboration. Service Component Architecture. Internet (http://www.osoa.org/display/Main/Service+Component+Architecture+Home), 2007.

[8] S. Majithia, D. W. Walker, and W. A. Gray. A framework for automated service composition in service-oriented architectures. In C. Bussler, J. Davies, D. Fensel, and R. Studer, editors, ESWS, volume 3053 of Lecture Notes in Computer Science, pages 269-283. Springer, 2004.

[9] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella. Automatic service composition based on behavioral descriptions. Int. J. Cooperative Inf. Syst., 14(4):333-376, 2005.

[10] R. Milner. Communication and concurrency. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[11] R. Milner. Communicating and mobile systems: the pi-calculus. Cambridge University Press, fifth edition, 2004.

[12] N. Milanovic and M. Malek. Current solutions for web service composition. IEEE Internet Computing, 8(6):51-59, 2004.

[13] R. Akkiraju, K. Verma, R. Goodwin, P. Doshi, and J. Lee. Executing Abstract Web Process Flows. In International Conference on Automated Planning and Scheduling (ICAPS), 2004.

[14] L. Zeng, A. H. H. Ngu, B. Benatallah, R. M. Podorozhny, and H. Lei. Dynamic composition and optimization of web services. Distributed and Parallel Databases, 24(1-3):45-72, 2008.

[15] D. Schuller, J. Eckert, A. Miede, S. Schulte, R. Steinmetz, QoS-Aware Service Composition for Complex Workflows, Proceedings of the 2010 Fifth International Conference on Internet and Web Applications and Services, 2010.

[16] M. Comuzzi, B. Pernici, A framework for QoS-based Web service contracting, ACM Trans. Web, Vol. 3, No. 3. (2009).

[17] S.-Y. Hwang, E.-P. Lim, C.-H. Lee, C.-H. Chen, "Dynamic Web Service Selection for Reliable Web Service Composition," IEEE Transactions on Services Computing, pp. 104-116, April-June, 2008.

[18] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, Q. Z. Sheng, Quality driven web services composition, Proceedings of the 12th international conference on World Wide Web (2003).

[19] A. Harrison, I. Wang, I. Taylor, and M. Shields, WS-RF Workflow in Triana, International Journal of High Performance Computing Applications (IJHPCA) Special Issue on Workflow Systems in Grid Environments, 2007.

[20] J. Kennedy and R. Eberhart, Particle swarm optimization, Proceedings of IEEE International Conference on Neural Networks, 1995.

[21] M. Clerc, Discrete particle swarm optimization - illustrated by the traveling salesman problem, New Optimization Techniques in Engineering, Springer, 2004.

[22] K.E. Parsopoulos, M.N. Vrahatis, Recent approaches to global optimization problems through particle swarm optimization, Natural Computing 1 (2002) 235–306.

[23] M. Clerc, J. Kennedy, The particle swarm explosion, stability, and convergence in a multidimensional complex space, IEEE Transactions on Evolutionary Computation 6 (2002) 58– 73.

[24] I.C. Trelea, The particle swarm optimization algorithm: convergence analysis and parameter selection, Information Processing Letters 85 (2003) 317– 325.

[25] Z. Michalewicz, D.B. Fogel, How to Solve It: Modern Heuristics, Springer-Verlag, 2002.

[26] J.-R. Zhanga, J. Zhanga, T.-M. Lokc, and M.R. Lyud, A hybrid particle swarm optimization–back-propagation algorithm for feedforward neural network training, Journal of Applied Mathematics and Computation, Volume 185, Issue 2, 15 February 2007, Pages 1026-1037.

[27] D. H. Kim, Ajith Abraham and K. Hirota, Hybrid Genetic: Particle Swarm Optimization Algorithm, Journal of Hybrid Evolutionary Algorithms in Studies in Computational Intelligence, 2007, Volume 75/2007, 147-170.

[28] S. N. Sivanandam, P. Visalakshi, A. Bhuvaneswari: Multiprocessor Scheduling Using Hybrid Particle Swarm Optimization with Dynamically Varying Inertia. IJCSA 4(3): 95-106 (2007)

[29] H.W. Kuhn, The Hungarian method for the assignment problem, Naval Research Logistics, 52(1), 1955.

[30] H.W. Kuhn, The hungarian method for solving the assignment problem, Naval Research Logistics Quarterly, 2:83, 1955.

[31] J. Munkres, Algorithms for the Assignment and Transportation Problems, Journal of the Society for Industrial and Applied Mathematics, 5:32, 1957.

[32] F. Bourgeois, J.C. Lassalle, An extension of the munkres algorithm for the assignment problem to rectangular matrices, Commun. ACM, 14(12), 1971.

[33] Wikipedia, Hungarian Algorithm, last retrieved December 2011, http://en.wikipedia.org/wiki/Hungarian_algorithm.

[34] K. Nedas, Munkres' (Hungarian) Algorithm, Java implementation, last retrieved on December 2011 from http://konstantinosnedas.com/dev/soft/munkres.htm.