Towards Network-Aware Composition of Big Data Services in the Cloud

Umar SHEHU  
Department of Computer Science  
and Technology University of  
Bedfordshire Luton, UK

Ghazanfar SAFDAR  
Department of Computer Science  
and Technology University of  
Bedfordshire Luton, UK

Gregory EPIPHANIOU  
Department of Computer Science  
and Technology University of  
Bedfordshire Luton, UK

Abstract—Several Big data services have been developed on the cloud to meet increasingly complex needs of users. Most times a single Big data service may not be capable in satisfying user requests. As a result, it has become necessary to aggregate services from different Big data providers together in order to execute the user's request. This in turn has posed a great challenge; how to optimally compose services from a given set of Big data providers without affecting if not optimizing Quality of Service (QoS). With the advent of cloud-based Big data applications composed of services spread across different network environments, QoS of the network has become important in determining the true performance of composite services. However current studies fail to consider the impact of QoS of network on composite service selection. Therefore a novel network-aware genetic algorithm is proposed to perform composition of Big data services in the cloud. The algorithm adopts an extended QoS model which separates QoS of network from service QoS. It also uses a novel network coordinate system in finding composite services that have low network latency without compromising service QoS. Results of evaluation indicate that the proposed approach finds low latency and QoS-optimal compositions when compared with current approaches.

Keywords—Big data; Service composition; QoS; Genetic Algorithm; Network latency; Cloud

I. INTRODUCTION

Service Oriented Computing (SOC) is a framework that allows for internet applications to be built by coupling web services together. In SOC, each web service represents a different functional aspect of a Service-oriented application (SOA) [32].

Web services are network-accessible objects that allow Big data vendors to build service-oriented Big data applications (SOA) which share business logic and application services with other vendors in order to meet growing consumer needs. A Big data service (BDS), also known as Big-data-as-a-service (BDaaS), is a data intensive web service that works on large scale unstructured or semi-structured datasets. They typically perform tasks such as data storage, processing, cleaning, extraction, modelling and visualization on large datasets. BDaaS consist mainly of three layers namely; the infrastructure layer, platform layer and application layer. The infrastructure layer provisions the physical resources required to process large datasets. The platform layer houses the operating systems and virtual machines that run BDaaS applications. The application layer represents the models and software used to process Big data.

Every BDS is characterized by the ability to provide some task as identified by its functional and non-functional attributes [27]. The functional attributes define what the service is capable of doing e.g. Microsoft Azure BDS [30] provides cloud-based machine learning framework for analyzing large scale datasets. Non-functional attributes on the other hand determine how well a service can perform a given task e.g. how long it will take Microsoft Azure BDS to respond to a user request. Non-functional attributes are commonly referred to as QoS (Quality of Service). Examples of service QoS attributes include response time, cost, reputation, etc. They are often used as criteria in selecting services suitable for a user request especially in situation where there is more than one service with similar functionality. For instance, a Microsoft Azure BDS having response time equal to 10 seconds will be more suitable to a user requiring prompt service response than a similar service such as Amazon AWS BDS with a response time of 30 seconds. Thus service QoS is used to differentiate services that are similar in terms of their functionalities. In SOC, functionally similar services are usually categorized in the same service group and referred to as candidate services.

A. QoS-Aware Web Service Composition

Recently, the ability of services to aggregate their functionalities has gained much attention. This is as a result of increased complexity of user requests. Simple user requests may require only a single BDS to be completed. However, as user requests take more complex forms that are beyond the capabilities of a single BDS, aggregating service abilities is necessary to carry out the request. This process is known as service composition. It combines services in order to build a composite service [8, 9] that is viewed by the user as a single service. Within a composite service each constituent BDS takes care of a specific functional aspect of the user’s request. For instance suppose a user issues a compound request like “Analyse e-books dataset” consisting of several sub requests at the task level such as Twitter feed analysis, Natural language processing and IOT device log analysis (as seen in Fig. 1). A single BDS is ill sufficient to satisfy the compound request, therefore services from different Big data vendors for each sub request will need to be discovered and aggregated together according to their QoS to complete the user request. QoS-aware service composition process is similar in principle to the behaviour of workflow management system [28] in which a workflow dictates how data should be processes. A workflow processes data by using different patterns that
transform data to an end result. Service composition uses similar patterns found in workflow systems. The patterns allow composition process to channel data flow from one BDS to another. Some major service composition patterns include sequence, parallel, exclusive choice and loop. Service composition process begins by breaking down a complex request into smaller functions or sub-requests organized according to one of several patterns. Depending on the pattern involved, service QoS are then orchestrated to determine QoS for a composite service. Usually there are several candidate services that exist within a service group that can execute a given functional aspect of the request.

Therefore choosing a service from each service group that maximizes the QoS of composite service while satisfying the user’s constraints has become a research problem. The problem is also known as an NP-Hard optimization problem [18]. The problem has been solved using several techniques such as Linear Integer Programming [13] and Dynamic Programming [11]. Although techniques based on genetic algorithms [18] are usually used in finding near optimal compositions in polynomial time.

B. Service Composition in the Cloud

More and more BDS are increasingly being deployed on the cloud with the purpose of allowing Internet users from around the globe to access their functionalities for analysing large datasets. For instance organizations such as Amazon and Microsoft offer public cloud services using Amazon Web Services (AWS) [31] and Windows Azure [30] cloud platforms respectively. These services are deployed on cloud data centres via virtual machines (VM) where consumers can access them from literally any part of the world. VMs enable the processing resources such as CPU, storage and network resources needed to properly run cloud-based BDS. Traditionally, service providers deploy their VMs across several cloud data centres located in different geographical areas to host their BDS. Hence, each user will experience different network performances depending on the geographical location of the hosted service. Thus, when a user tries to invoke a composite service with candidate services spanning different cloud locations, the composition may not be able to deliver on the network performance needs of the user even if it has optimal service QoS. This is because the optimal service QoS only represents application level performance of a composite service but it does not account for its network performance. The impact of the network is usually quantified using a metric such as network latency [22]. The effect of network latency on application performance is noticeable in cloud environments where there is high degree of service distribution. Despite this, current studies do not separate QoS of network from service QoS. Hence, they may produce compositions that have sub-optimal performance when invoked by the user. An example is illustrated in Fig. 2. The example shows several BDS deployed on different clouds. Assuming each cloud consists of two or more BDS and is separated from other clouds by different round trip times (RTT). Also assuming a user request consists of a sequence pattern of the three tasks (t1, t2, and t3) in Fig. 1, with each task having a set of candidate services and their respective QoS scores for cost (C), response time (RT) and execution time (ET). Current approaches will ordinarily pick the QoS optimal composite service (highlighted using bold boxes in Fig. 3) consisting of services (i.e. S_A1-S_B1-S_C1) with respect to cost, execution time and response time.

BDS having shorter RTT will incur lower latency than those further away from each other. Therefore user A may experience low network latency for composite service |S_{A1}-S_{B1}-S_{C1}| (i.e. end-to-end network latency for |S_{A1}-S_{B1}-S_{C1}| is 400ms + 100ms + 54ms + 500ms = 1054ms), while user B experiences high network latency because of larger RTT (i.e.
Several studies have dealt with service composition while considering the impact of QoS of the network. Authors in [3] present a network-based service composition technique for component services in large scale overlay networks. Similarly, another study in [4] introduces network awareness in composing domain services in multi-domain networks. The authors try to optimize delay and available bandwidth. However, these studies do not consider service composition in the context of services in the cloud. A recent approach [2] develop a service composition technique that minimizes network latency of composite services in the Cloud. The authors use a network model based on Euclidean distance technique to estimate latency of composite services. Their work is similar with this study in that they consider network latency. The main difference is that they only consider QoS of network while this work considers QoS of network alongside service QoS objectives.

C. Network Coordinate System

Network coordinate systems (NCS) are used to estimate latency between nodes in a network [2]. Their purpose is to reduce the delay observed from sending physical round trip time (RTT) packets between nodes across the network path. They operate by predicting RTT measurements for a fraction of nodes on the network path using techniques such Euclidean distance estimation (EDE) [5, 16, 17] and matrix factorization (MF) [21]. EDE embed network distances between nodes as metric spaces where known network distances (RTT) are mapped into a two dimensional Euclidean space in order to predict unknown network distances. EDE is however susceptible to triangle inequality [21] which leads to inaccurate estimates. MF on the other hand estimate unmeasured network distances by factorizing distance matrix consisting of both known and unknown RTT values using mathematical concepts such as gradient descent [19]. MF does not use metric spaces and so is resistant to triangle inequality and produces more accurate than EDE. Current EDE and MF models adopt a centralized approach towards RTT estimation. The approach usually involves using a central server to collectively predict RTT values for all the nodes in the network path. This means that if one RTT value is inaccurately estimated, then the accuracy of other RTT values could be negatively affected. In this work, the problem is avoided by adopting a novel decentralized MF approach within the network model where each BDS takes charge of predicting RTT with its neighboring services independently of other services on the cloud network.

III. PROPOSED APPROACH

A. Problem Formulation

The problem can be described as follows:

Given a user request T that will require a set of tasks \( t_i \) to \( t_n \),

\[ T = \{t_1, t_2, \ldots, t_n\} \]

Where \( n \) is the number of tasks to complete user request.
Each task is assigned a service group \( (S) \) which defines a set of candidate services \( (s_{ij}) \) capable of performing the given task (as seen in Fig. 4).

\[
S_i = \{s_{i1}, s_{i2}, \ldots, s_{ik} \}, \quad \forall i \in [1..n], \forall j \in [1..k]
\]

Where \( k_j \) is the number of candidate services in the \( i \)-th service group.

For each task, only one candidate service within its service group can be bound to the task \( t_i \) to form a composite service \( C \).

\[
\begin{array}{ccc}
\{ t_1 \} & \{ t_2 \} & \{ t_n \} \\
\{ s_{i1} \} & \{ s_{i2} \} & \{ s_{in} \} \\
\{ s_{i2} \} & \{ s_{i3} \} & \{ s_{nk} \} \\
\cdots & \cdots & \cdots \\
\{ s_{ik} \} & \{ s_{ik} \} & \{ s_{ik} \}
\end{array}
\]

Fig. 4. Classification of candidate services into service groups and tasks

A composite service can be formed from the aggregation of candidate services:

\[
C = \{s_{ij}, s_{ij}, \ldots, s_{ij}\}, \quad \forall j \in [1..k], \forall i \in [1..n]
\]

Where \( s_{ij} \) is the BDS bound to its service group \( S_i \).

Also, given a set of QoS objectives (cost, execution time, response time and network latency) that need to be optimized, the end-to-end QoS value of a composite service \((Q(C))\) is calculated by combining individual QoS values of its services (one per task) based on the following expressions.

In order to determine end-to-end cost of composite service, cost for each service \((P(s_{ij}))\) are combined

\[
Q_p(C) = \sum_{i=1}^{n} P(s_{ij}) \quad (1)
\]

Similarly, both end-to-end response time \((RT(s_{ij}))\) and execution time \((ET(s_{ij}))\) are aggregated respectively

\[
Q_{RT}(C) = \sum_{i=1}^{n} RT(s_{ij}) \quad (2)
\]

\[
Q_{ET}(C) = \sum_{i=1}^{n} ET(s_{ij}) \quad (3)
\]

As for end-to-end network latency, RTT values are combined between each service in a given composite service

\[
Q_{NL}(C) = \sum_{i=1}^{n} NL(s_{ij}, s_{i+1,j}) \quad (4)
\]

Where \( NL(s_{ij}, s_{i+1,j}) \) represents the round trip time between each BDS in the cloud. \( Q_p, Q_{RT}, Q_{ET} \) and \( Q_{NL} \) represent end-to-end cost, response time, execution time and network latency of a composite service respectively.

Given weights \( w^p, w^RT, \) \( w^ET \) \( w^NL \) which represent relative importance of QoS objectives from the user’s perspective. Where,

\[
\sum_{m=1}^{4} w_m = 1 \quad (5)
\]

QoS objectives are normalized into fitness values using the expressions in Equations (6) and (7). Cost, response time and execution time are computed thus

\[
F_m(C) = \sum_{i=1}^{n} \left( \frac{\max_m(S_i) - q_m(s_{ij})}{\max_m(S_i) - \min_m(S_i)} \right) \times w^m_m \quad (6)
\]

\[
\forall j \in [1..k]
\]

Network latency fitness value for composite service \( (F_{NL}) \) is determined by an expression in Equation (6) which normalizes the end-to-end network latency QoS \( (Q_{NL}) \).

\[
F_{NL}(C) = \frac{Q_{NL}(C)}{H} \times w^NL \quad (7)
\]

Where \( H \) is a constant which normalizes value of \( Q_{NL}(C) \) in the range of \([0,1]\).

The research problem becomes a constrained multi-objective optimization problem where the aim is to find a set of composite services with near-optimal fitness values,

\[
C_{best} = \text{Min} \left[ F(C) \right]
\]

Subject to:

- Selection constraint: Only one candidate service can be selected per service group.

- Minimum \( (q^\min_m) \) and maximum \( (d^\max_m) \) QoS constraints:

\[
\forall Q_p \in [q^\min_p, q^\max_p], \forall Q_{RT} \in [q^\min_{RT}, q^\max_{RT}], \quad \forall Q_{ET} \in [q^\min_{ET}, q^\max_{ET}]
\]

\[
B. \text{ Network Model}
\]

In this study, a network model for estimating the RTT between BDS deployed on the cloud is adopted. The network model is composed of network coordinate system (NCS) based on Matrix factorization called LADMF (Learning-based Decentralized Matrix Factorization). Traditional MF techniques measure RTT \((d_{h_{i-s_{jk}}})\) between a BDS and a subset of neighbors to build distance matrix \( D \). These
measurements are then used to predict RTT values \( d^*_{x_i-y_{old}} \) for non-neighbouring services as seen in Fig. 5. In mathematical terms, standard MF finds estimates of row matrix \( X \) and transposed column matrix \( Y \) that minimize the difference \( \varepsilon \) between measured RTT values in \( D \) and in estimated values in matrix \( D_{new} \):

\[
\min[\varepsilon] 
\]

Where \( \varepsilon \) is the latency prediction error;

\[
\varepsilon = (D - D_{new})^2 \tag{9}
\]

Also \( D_{new} \) is expressed as;

\[
D_{new} = X * Y^T \tag{10}
\]

Where \( X \) and \( Y \) are positional coordinates of all BDS on a given cloud network.

the standard MF technique is modified by adding learning automata concepts in order to further improve prediction accuracy of the estimation process. Instead of constructing a collective matrix \( (D_{new}) \) for all RTT estimates, LADMF decentralizes the process by allowing each BDS to estimate its own RTT values irrespective of other services. This is achieved by encoding each service as a learning automaton (LA) \[5\]. LA converts Equations (10) and (9) into Equations (11) and (12) respectively,

\[
D_{ij_{new}} = X_i * Y_j^T \tag{11}
\]

\[
\varepsilon = \left[ (D_{ij} - D_{ij_{new}})^2 \right] \tag{12}
\]

Where \( X_i \) is positional coordinate of \( i \)-th service, \( Y_j \) is positional coordinate of \( j \)-th neighbouring service, while \( D_{ij} \) is the RTT between services \( i \) and \( j \).

The effect is that each BDS will control their own path to RTT estimation without influencing estimation path of other services. Hence an inaccurate estimation of one service coordinate will not affect accuracy of other service coordinates.

In LADMF \( X_i \) and \( Y_j \) are encoded with additional LA parameters as seen in Fig. 6.

\[
X_i = \langle x_{i1}, \ldots, x_{i3} \rangle \quad Y_j^T = \langle y_{j1}, \ldots, y_{j3} \rangle
\]

Fig. 6. Encoding of position vectors with LA parameters

Where

- \( \alpha \) represents two alternative update strategies (\( \alpha_1 \) and \( \alpha_2 \)) employed in updating position coordinates in \( X_i \) and \( Y_j \):

\[
\begin{align*}
\alpha_1 &> X_{i_{(new)}} = D_{ij}Y_j(Y_j^TY_j + (\Omega + J_1))^{-1} \\
Y_{j_{(new)}} & = D_{ij}X_i(X_i^TX_i + (\Omega + J_2))^{-1} \\
\alpha_2 &> X_{i_{(new)}} = D_{ij}Y_j(Y_j^TY_j + (\Omega - J_1))^{-1} \\
Y_{j_{(new)}} & = D_{ij}X_i(X_i^TX_i + (\Omega - J_2))^{-1}
\end{align*}
\]

Also

- \( \Omega \) - Regularization parameter that controls speed of update
- \( J_1 \) and \( J_2 \) are constants
- \( I \) - Identity matrix
- \( \beta \) represents feedback for every action in \( \alpha \), \( \beta = \{\beta_{\alpha_1}, \beta_{\alpha_2}\} \)
- \( P_{\alpha} \) is action probability which is determined from feedback of estimation error.

If feedback for action \( \alpha_1 \) is good \( (\beta_{\alpha_1} = 0 \) i.e. \( \varepsilon \) is improved) then action probability \( P_{\alpha_1} \) is rewarded while \( P_{\alpha_2} \) is penalized,

\[
\begin{cases}
P_{\alpha_1_{(new)}} = P_{\alpha_1} + \epsilon(1-P_{\alpha_1}) & c=0.5, \\
P_{\alpha_2_{(new)}} = P_{\alpha_2} - \epsilon(1-P_{\alpha_2}) & e=0.005*\epsilon
\end{cases}
\]

Else if feedback is bad \( (\beta_{\alpha_1} = 1 \) i.e. \( \varepsilon \) is not improved) then reverse is the case,

\[
\begin{cases}
P_{\alpha_1_{(new)}} = P_{\alpha_1} + \epsilon(1-P_{\alpha_1}) & c=0.5, \\
P_{\alpha_2_{(new)}} = P_{\alpha_2} - \epsilon(1-P_{\alpha_2}) & e=0.005*\epsilon
\end{cases}
\]

Actions are evaluated and assigned probabilities based on error feedback which in this case is the estimation error (uniformly distributed)}
min(ε). The action with the highest probability is selected as the next action. The process is continued until the estimation error is minimized. LADMF algorithm is outlined in Algorithm 1. Afterwards, estimated RTT values are aggregated to determine end-to-end network latency for a composite service via Equation (4).

Algorithm 1 LADMF Algorithm

Input: D, maxIter, L
Output: Dnew
1: \( [X, Y] \) = function LADMF(D)
2: for(i = 1: maxIter)
3: \( X \leftarrow \text{rand}(x) \)
4: \( Y \leftarrow \text{rand}(y) \)
5: \( \varepsilon \leftarrow w[D-(X * Y)]^2 \)
6: if (\( \varepsilon \) is minimised)
7: \( \text{Dnew} \leftarrow X * Y \)
8: return
9: endif
10: endfor
11: return
12: endfor
13: }

C. Network-Aware Service Composition Algorithm

A novel network-aware service composition technique based on non-dominated sort genetic algorithm (NSGA) is presented. When applying genetic algorithm to service composition problem, each genome represents a possible composite service and is encoded in form of array of numbers or genes, each gene in turn represents a task and can be assigned to any one of its candidate services (as seen in Fig. 7). State of the art NSGA initiates optimization process by building an initial generation of genomes then sorts individuals according to their fitness value and crowding distance. The best individuals are placed in a mating pool where they are altered by crossover and mutation operators to generate children that will populate subsequent generations. The whole process is repeated until optimization is reached. The state of the art NSGA algorithm is enhanced in order to be able to solve research problem. The improved algorithm called INSGA is described step by step as follows:

Step 1. Initialization of Population. INSGA starts by randomly generating an initial population from the BDS that are part of the cloud. In order for this to be achieved, every service is first encoded as a two digit integer value. For example in Fig. 8, a BDS is encoded as "33" is the 3rd candidate service capable of executing task 3. In the next step only one candidate service is arbitrarily selected per task.

BDS QoS scores are then randomly initialized within their boundary constraints. With the aid of LADMF algorithm, the QoS scores are normalized and aggregated into values representative of composite service end-to-end cost, response time, execution time and network latency respectively.

Step 2. Ranking and Sorting. INSGA uses a non-dominated sorting technique that ranks individuals into different fronts according to the degree that they dominate other individuals in the population. A composite service \( C_i \) perfectly dominates another composite service \( C_j \) if all four fitness values of \( C_i \) are lower than the fitness values of \( C_j \). Therefore \( C_i \) will be placed in a higher rank (front) than \( C_j \).

For each front, individuals are sorted in ascending order according to the magnitude of their fitness. This is used to establish the crowding distance (CD) which indicates the Euclidean distance between individual in the fitness value space. CD for a given composite service \( C_i \) is expressed as:

\[
CD(C_i) = \frac{F(C_{i+1}) - F(C_{i-1})}{F_{\text{max}} - F_{\text{min}}} \tag{16}
\]

Where

- \( CD(C_i) \) is the crowding distance for the \( i \)-th individual.
- \( F(C_{i+1}) \) represents the fitness value of individual succeeding \( i \)-th individual.
- \( F(C_{i-1}) \) represents the fitness value of individual preceding \( i \)-th individual.
- \( F_{\text{max}} \) and \( F_{\text{min}} \) represent the maximum and minimum fitness values in population

Step 3. Tournament Selection. A tournament selection of the best individuals that meet the user’s satisfaction constraint is achieved to determine parents who will take part in crossover operation. The selection process ensures that only individuals with best fitness, rank and do not violate user constraint are selected for crossover operation.

Step 4. Crossover Operation. Crossover operation combines any two parents into offspring (children) that are quite different from their parents and can have superior
properties of both parents. Traditional crossover operation picks arbitrary cut points where genes around cut points of one parent are replaced with genes of another parent to construct a set of children. INSGA employs a novel two-point crossover which cuts parents at two non-random cut points. The two cut points (one per parent) are chosen from points on each parent where average network latency is high. In order to determine which point on a parent constitutes poor average latency, every BDS assigned an average latency score \( A_L \) which is the arithmetic sum of RTT values over all outgoing paths divided by the number of outgoing paths from a given service.

\[
A_L(s) = \frac{1}{|G|} \sum_{g \in G} Q_{NL}(g) \quad (17)
\]

Where \( A_L(s) \) represents average latency score in milliseconds (ms) for service \( s \), \( G \) is number of outgoing paths from \( s \), and \( Q_{NL}(g) \) is RTT value for a given path.

Once average latency scores are known, the crossover operator selects a cut point from each parent where \( A_L \) is maximum. After the cut points are known then the genes around those points are interchanged between both parents. This ensures that genes having highest \( A_L \) are interchanged with genes having lower \( A_L \). Fig. 9 depicts how crossover operation is performed.

When cut points 1 and 2 are the same for both parents then the crossover operation translates to a single point crossover. The impact of the crossover operator is that children produced are low latency versions of their parents as demonstrated by the results.

*Step 5. Mutation Operation.* The function of mutation operation is to adjust a parent into new offspring that closely resemble its parent with the aim of further improving parent fitness values and discourage trapping into local optima. The standard mutation operator adjusts parents by using a uniform distribution index (DI) [23]. DI controls degree of similarity between parents and their children. The value for DI influences the diversity of offsprings in the population. A new mutation operation is presented. The operator uses a variable distribution index whose value depends on a parent’s crowding distance and fitness value for network latency. Each parent is going to be mutated according to the value of its distribution index which is computed using the following expression:

\[
mum_{par} = \left\{ \frac{H}{F_{NL}(par_i) + (1 - CD(par_i))} \right\} \times CD(par_i) \quad (18)
\]

Where

- \( mum_{par} \) is the distribution index for the parent.
- \( F_{NL}(par_i) \) represents the parent’s fitness value for network latency.
- \( CD(par_i) \) indicates the parent’s crowding distance.
- \( H \) is a constant.

The expression in Equation (18) will force a strong mutation for poor quality parents and a weak mutation for good quality parents. A large value for \( mum_{par} \) will indicate parent has good fitness and crowding distance therefore offspring’s genes will closely resemble the parent (i.e. weak mutation), while a small value for \( mum_{par} \) indicates parent has poor fitness and crowding distance hence genes of offspring will differ greatly with the parent (i.e. strong mutation). This will ultimately improve the population diversity of new offspring and also increase the likelihood of finding the global solution. After mutation operation is performed, parents are replaced by newly formed off springs and the whole process is repeated until maximum number of generation is reached. INSGA algorithm is outlined in Algorithm 2 while the unique crossover and mutation operators are outlined in Algorithm 3 and 4 respectively.

**Algorithm 2 INSGA Algorithm**

| Input: | D, g, n, Q, h, max_iter, no_states, state, actions_prob, ipenv, w, J1, J2 |
| Output: | pop |

1: Set environment parameters
2: pop ← Randomly generate population
3: \( P \) ← Randomly generate QoS values of solutions
4: \( pop(Q, f) \) ← Determine end-to-end QoS and fitness of solutions
5: pop ← Perform non-dominated sort (pop)
6: pop ← LADM (Input)
7: While (gen ≠ maxgen) do
8: \{ 
9: pop ← tournament selection (pop)
10: pop ← Crossover (pop)
11: pop ← Perform non-dominated sort (pop)
12: child_pop ← Mutation (pop)
13: combination_pop ← pop + child_pop
14: combination_pop ← Perform non dominated sort (combination_pop)
15: pop ← replacement (combination_pop)
16: endWhile
17: \}
between ironment. The dataset contains demonstrate the efficiency of INSGA, because it is expensive to implement a physically population diversity index against fitness, because it is expensive to implement a physically population diversity index against fitness, because it is expensive to implement a physically population diversity index against fitness, because it is expensive to implement a physically population diversity index against fitness, because it is expensive to implement a physically population diversity index against fitness, because it is expensive to implement a physically population diversity index against fitness, because it is expensive to implement a physically population diversity index against fitness. This result shows that improvements in fitness, latency and population spread can be attributed to the proposed mutation and crossover operators.

---

**Algorithm 3 INSGA Crossover operation**

**Input:** pop
**Output:** Child
1: For(i = 1 to popsize)
2: |
3: Randomly pick Parent1 and Parent 2 from pop
4: Compute Average latency \( \bar{A} \) of Parent 1 and Parent 2
5: \( index1 \) ← Find cut point of Parent 1 with poorest latency
6: \( index2 \) ← Find cut point of Parent 2 with poorest latency
7: \{Child 1, Child 2\} ← Crossover genes for each parent around \( index 1 \) and \( index 2 \)
8: \{Child 1, Child 2\} ← Determine end-to-end QoS and fitness of children
9: Child ← Add Child 1 and Child 2 in the child population.
10: endFor
11: 

**Algorithm 4 INSGA Mutation operation**

**Input:** pop
**Output:** Child
1: For(i = 1 to popsize)
2: |
3: Compute Dist. Index of pop(i) according to Equation (18)
4: Child(i) ← Mutate genes of pop(i) according to DI
5: Child(i) ← Determine end-to-end QoS and fitness of child
6: endFor
7: 

---

### IV. Evaluation

**A. Setup**

Experiments were run on a machine with Intel Core i7 CPU (3.8GHz) and with 8GB memory. All the algorithms and experiments are implemented in MATLAB 2013. A cloud network of BDS is simulated using planet lab meridian dataset [7] to provide RTT measurements between BDS. The dataset is chosen because it is expensive to implement a physically large cloud environment. The dataset contains symmetric round trip time (RTT) measurements between 1740 peer-to-peer nodes. Also, a test workflow is generated and will be used to evaluate INSGA algorithm. In the workflow, a set of thirteen tasks \( t_1 \) to \( t_{13} \) is defined. For each task, it is assume that each service group has equal number of candidate services for the sake of simplicity. The experiment is performed with 20 candidate services per task to simulate a large BDS cloud network.

**B. Results and Discussion**

To demonstrate the efficiency of INSGA, its fitness latency and population diversity are compared against other traditional algorithms such as Particle swarm optimization (PSO) [26] and Genetic algorithms N-NSGA [25] and SNSGA [24] in different environmental situations such as variations in number of tasks, candidate services and distribution index. Given the probabilistic nature of the test algorithms, each algorithm is run 50 times to obtain average values for fitness, latency and standard deviation which is often used to measure diversity of population.

- **a) Impact of Distribution Index**
  
  In this experiment, an evaluation is done to determine the impact of distribution index on average fitness and population diversity of composite services. Here, the population size and maximum generation are set as 200 with network size of 260 services. In Fig. 10 (a) (b) and (c), it is observed that INSGA finds solutions with better fitness, latency diversity than N-NSGA and N-NSGA80. INSGA also avoids trapping in local optima while converging after 140 generations. This result shows that improvements in fitness, latency and population spread can be attributed to the proposed mutation and crossover operators.

---

(a) Graph indicating effect of distribution index on fitness

(b) Graph showing effect of distribution index on latency

(c) Graph showing effect of distribution index on population diversity

---

**b) Size of Candidate Service per Task**

In this experiment, the number of candidate services per task is increased from 20 to 50 and evaluate the impact on network latency, fitness, computation time and standard deviation of population. In Fig. 11(a) and (b), it is noticed that an increase in size of candidate services may ultimately lead to better quality solutions for all test algorithms with the exception of PSO whose quality worsens. It can also be seen...
that INSGA finds solutions with the best fitness value and network latency when compared to the other algorithms. In Fig. 11(c) the computation times of all four algorithms are compared. It is observe that only INSGA has the highest computation time. This is as a result of the computational overhead generated by the network model. PSO has the lowest computation time which is about one third of INSGA's time. Also computation times for both N-NSGA, PSO and INSGA increase slightly with the number of candidate services except for S-NSGA whose computation time remains largely unchanged. Fig. 11(d) shows that increasing candidate service size doesn’t influence the diversity of population for N-NSGA, PSO and S-NSGA. But in INSGA, standard deviation is improved slightly. Also, PSO shows the worst standard deviation while INSGA has the best standard deviation amongst the test algorithms.

(c) Size of Tasks
In this experiment the number of tasks are varied from 13 to 40 then the impact of fitness, network latency, computation time and standard deviation on the algorithms are determined. In Fig. 12 (a) and (b), it is observed that quality of fitness and network latency degrades with size of tasks for all test algorithms. INSGA is seen to produce the best quality solutions in terms of fitness and latency (tied with N-NSGA) while PSO produces worst quality of solutions. In Fig. 12 (c) a pattern similar to Fig. 11 (c) is observed, the only difference noticed is that computation time peaks at higher values when compared to graph in Fig. 12 (c). Lastly Fig. 12 (d) shows that population diversity increases linearly with size of tasks.

(a) Graph showing impact of number of candidate services on fitness

(b) Effect of number of candidate services on network latency

(c) Effect of number of candidate services per task on computation time

(d) Effect of number of candidate services per task on population diversity

Fig. 11. Plot of candidate service size against fitness, network latency, computation time and standard deviation
art service composition algorithms, results show that INSGA average latency is maximum while the mutation operator crossover operator non response time execution time and network latency QoS. It consists of non-dominated sort genetic algorithm called INSGA.

The aim of INSGA is to multi-objectively optimize cost, response time execution time and network latency QoS. INSGA uses a custom crossover and mutation operator. The crossover operator non-randomly picks two cut points where average latency is maximum while the mutation operator varies distribution index as a function of crowding distance and network latency. When compared with other state of the art service composition algorithms, results show that INSGA finds better quality solutions in terms of fitness, network latency and global search ability as indicated by its standard deviation.

REFERENCES


[17] Ng, T.S.E; Zhang, H.A.;”A Network Positioning system for the Internet,” in Proc USENIX ATLAS on, vol., pp., 2004


[19] Yongjun Liao; Wei Du; Geurts, P.; Leduc, G., "DMFSGD: A Decentralized Matrix Factorization Algorithm for Network Distance


