Optimizing QoS-aware services composition for concurrent processes in dynamic resource-constrained environments

Yuanhong Shen, Xiaohu Yang, Ye Wang, Zhen Ye
School of Computer Science and Technology, Zhejiang University, Hangzhou, 310027, China
{yhshen,yangxh,ai_4869,yezhen}@zju.edu.cn

Abstract—QoS-aware service composition intends to integrate services from different providers and maximize the global QoS in order to increase the user’s satisfaction degree while subjecting to dynamic context constraints. Current composition approaches only focus on optimizing a single process to maximize the satisfaction degree for one party. When multiple processes are performed concurrently by their selfish users in a dynamic resource-constrained environment, new issues will arise, i.e., undesirable competition for service resources, extra waiting and frequent change of contexts. To address these issues, this paper aims to optimize QoS-aware services composition for multiple selfish users if the communication among users is allowed. Firstly, we propose an extensional QoS-aware service selection model for each process. Then based on this model, we present fault handling mechanisms before and during the execution of concurrent composite services for concurrent processes based on a multi-issue negotiation protocol among agents, and an adaptive context-aware service re-selection mechanism for adjusting the service execution plan for each running composite service in the dynamic resource-constrained environment. Comparative experiments reveal our approach facilitates to increase the average satisfaction degree, reduce the average waiting time of multiple users, and make the satisfaction degrees among multiple users more evenly distributed in the dynamic resource-constrained environment.

Keywords—service composition; QoS; multi-issue negotiation; resource-constrained; context

I. INTRODUCTION

In service-oriented environments, service composition is the process that integrates existing Web services provided by different organizations. In the composition process, the functional capability and Quality of Service (QoS) properties [1] of the selected Web services (concrete candidate services) for the tasks (abstract services) need to be considered to ensure functional and QoS constraints on the composite service and meet users’ satisfaction degrees.

In the context of dynamic service environment, device, policy, user constraints and QoS requirements may change, new services may be deployed, while old ones may become unavailable, or existing ones may change their QoS values [2, 28]. All of these situations may result in the failure of the current service composition plan or its aggregated global QoS values degrading from users’ expectations. Therefore, service composition needs to adapt to the dynamic context at runtime to ensure the dependability and flexibility of the distributed SOA systems.

Many researchers have carried out on adaptive QoS-aware Web service compositions [2, 4-6]. However, almost all of current methods focus on optimizing a single process to maximize the satisfaction degree for one party. The limitation of computing capability of the resources on devices for the computing requirement of each running Web service [7, 16] (‘device resources’ for short), and the limitation of optimal services [22, 27] are often neglected. In many reality scenarios, such as in mobile computing environments [8], device resources are limited [8] for running multiple services concurrently. When multiple processes are performed concurrently by their selfish users in a dynamic resource-constrained environment, the following issues will arise which cannot be predicted by a single user:

• Undesirable competition for service resources: there may be inadequate candidate services for each task (‘service resources’ for short) to act as replacements to increase the availability for all the composite services. Undesirable competition for service resources may exist among concurrent tasks, because some tasks may occupy unused services with good performances while other tasks cannot occupy any feasible services.

• Extra waiting: we focus on the extra waiting caused by inadequacy of device resources. Suppose service $S_a$ in composite service Com$S_a$ is planned to run on device $d_s$, at time $t_f$ for 100s and $S_b$ in composite service Com$S_b$ is planned to run on device $d_s$ at $t_f+10s$ for 10s, and there are not enough device resources for performing both $S_a$ and $S_b$ in $d_s$ concurrently. If there is no collaboration between Com$S_a$ and Com$S_b$, Com$S_b$ has to wait for 90s until $S_a$ finishes due to inadequate device resources. However, if $S_a$’s device resource requirements can be known by Com$S_b$ ahead of time, the overall waiting time of Com$S_a$ and Com$S_b$ may be reduced to 10s, and there exists an extra waiting time (80s) for Com$S_b$. Therefore, the overall satisfaction degree of multiple users requesting different composite services will be reduced, and the satisfaction degrees among multiple users will vary significantly.

• Frequent change of contexts: when multiple composite services are running concurrently in a dynamic resource constrained environment, the execution contexts for each composite service will be more unstable due to variations of available device resources. Therefore, current composition mechanisms need to be adaptive to the variations of
runtime context to improve the performance of the composite services.

This paper aims to optimize QoS-aware services composition for concurrent processes in a dynamic resource-constrained environment under the situation that the communication among the users is allowed, where each user requests for a composite service. The contributions of this paper are four-fold: 1) a framework about services composition for multiple concurrent processes in a resource-constrained environment; 2) an extensional QoS-aware service selection model for each process; 3) fault handling mechanisms for concurrent composite services for concurrent processes based on a multi-issue negotiation protocol among agents, addressing the issues of undesirable competition for service resources and extra waiting before and during the execution of concurrent composite services; 4) an adaptive service re-selection mechanism for adjusting the service execution plan for each running composite service in the dynamic resource-constrained environment. Comparative experiments reveal our approach can help to increase the average satisfaction degree, reduce the average waiting time of multiple users, and make the satisfaction degrees among multiple users more even distributed in a dynamic resource-constrained context sensitive environment.

II. QOS-AWARE SERVICES COMPOSITION FRAMEWORK FOR MULTIPLE CONCURRENT PROCESSES

Our models and methods for optimizing QoS-aware services composition for concurrent processes are all contained in an integrated QoS-aware service composition framework. Before introducing the framework, we first introduce some basic concepts as follows: 1) service plan (SP) refers to the required process specification proposed by a user, composed by a set of tasks, structure information [10] (sequence, fork, choice and loop structures) and a set of properties (e.g., execution probabilities of conditional branches, loop constraints, and global and local constraints on quality dimensions [5]); 2) composite service represents the acquired Web services integrated by other candidate (Web) services for a SP; 3) negotiation agent (NA) and coordinator agent (CA) represent the pieces of software that autonomously act to carry out the fault handling mechanism if a failure occurs in one composite service (the detail is described in Section IV). Each NA represents a selfish user requesting a SP: 4) service contexts (collected from distributed service providers) represent the functionalities, QoS values, and device information of all the candidate services; 5) device contexts (collected from distributed device providers) represent the inner device resource constraints and resource constraints between different devices [2] (all candidate services are deployed on the devices); 6) other QoS related contexts represent business rules and other constraint information that influences the QoS-aware selection of candidate services for the SP. All context information is managed and monitored by each context manager for each SP.

As shown in Fig. 1, the work procedure of our services composition is as follows: 1) each user defines a SP, 2) the Context Manager obtains a list of candidate services (with equivalent function but different QoS values) and the related context information for each SP, 3) each service selection module competes for an optimal composite service for each SP, 4) before the execution of each acquired composite service, the NAs communicate with each other to handle faults if the composite service for one SP cannot meet the user’s requirement, 5) each execution engine in the framework executes the composite service by invoking the selected Web services in the composite service. If one composite service suspends in the process of execution, all NAs also negotiate to determine better service execution strategies to improve overall satisfaction degrees of all users. In addition, each running composite service can adapt to the changing environment by the strategy adaption module.

III. EXTENSIONAL QOS-AWARE SERVICE SELECTION MODEL

In this section, we briefly introduce an extensional QoS-aware service selection model for each concurrent process, which is the basis of the fault handling mechanisms for concurrent composite services and the adaptive context-aware service re-selection mechanism in the following sections. Considering the length of the paper, the derivation processes of some formulas are omitted.

A. Extensional QoS model

QoS provides non-functional characteristics for the optimal Web service selection [3]. Based on the investigation of previous works [1, 2, 3, 5, 28], we take the QoS (q) including cost (c), latency (l), successful execution rate (e), and reputation (r) into account for each candidate service. We assume all QoS information can be collected or predicted before service executes by service providers and users. The QoS value of the composite service, which aggregates the QoS of each task, can be calculated according to different structures [3].

For better resource conservation in dynamic resource-constrained environment, we apply the sequential strategy [11] for each task in a process. In the sequential strategy,
When a replica fails due to service faults, the strategy will determine whether to retry the current replica or invoke a new replica. After investigations on [11] and other related works, and considering the inadequacy of device resources as our major issue, we divide service faults into resource-dependent faults and resource-independent faults: resource-dependent faults represent the faults caused by inadequate device resources, while resource-independent faults are caused by other reasons. We assume the resource-dependent fault rate of a replica \( S_{ij} \) is \( f_d(S_{ij}) \) and the resource-dependent fault rate of the service \( S_i \) is \( f_d(S_i) \). Thus, the fault rate of \( S_{ij} \): 
\[
f(S_{ij}) = f_d(S_{ij}) + f_d(S_i) - f_d(S_{ij})f_d(S_i)
\]
We assume the resource conflicts can be detected by the users. When a replica fails due to resource conflicts, a new replica will be invoked, otherwise, the replica may be re-tried. We assume the maximal number of retries for the \( m \) replica is \( m_i \). If all replicas for \( Ti \) fail, \( Si \) wait for released device resources for its deployment and invocation. Based on the sequential strategy for each task, the conventional QoS model is extended as follows:

The fault rate of the first \( n \) replicas for \( Ti \): 
\[
f(T_i, n) = \prod_{j=1}^{n} \sum_{l=0}^{m_i-1} \left( f_d(S_{ij})(1-f_d(S_i))^{l}f_d(S_i)\left(1-(1-f_d(S_i))^{m_i-1}\right)\right)
\]
\[
f_d(S_{ij})(1-f_d(S_i))^{l}f_d(S_i)\left(1-(1-f_d(S_i))^{m_i-1}\right)\right)
\]
The execution of replicas for \( Ti \) will not continue until the execution of the \( k \)th \((1\leq k\leq n)\) replica is successful in its \( p \)th \((1\leq p\leq m_i)\) time for \( Ti \) or all the \( n \) replicas fail for \( Ti \). The successful execution on the \( k \)th replica in the \( p \)th time for \( Ti \) is represented by \( A(T_i, k, p) \). The rate of \( A(T_i, k, p) \) can be formulated using Eq. (2):
\[
A(T_i, k, p) = \prod_{j=1}^{n} \sum_{l=0}^{m_i-1} \left( f_d(S_{ij})(1-f_d(S_i))^{l}f_d(S_i)\left(1-(1-f_d(S_i))^{m_i-1}\right)\right)
\]
\[
= \prod_{j=1}^{n} \sum_{l=0}^{m_i-1} \left( f_d(S_{ij})(1-f_d(S_i))^{l}f_d(S_i)\left(1-(1-f_d(S_i))^{m_i-1}\right)\right)
\]
The successful execution on the \( k \)th replica for \( Ti \) is represented by \( A(T_i, k) \). The rate of \( A(T_i, k) \) is calculated using Eq. (3):
\[
A(T_i, k) = \sum_{j=1}^{n} A(T_i, k, j)
\]
The expected QoS \((c, l, e, r)\) for \( Ti \) (represented by \( T_i.c, T_i.l, T_i.e, T_i.r \)) is integrated into the QoS of \( S_i \) (represented by \( S_i.c, S_i.l, S_i.e, S_i.r \)). The expected QoS values for \( Ti \) can be calculated using Eq. (4)-(7):
\[
T_i.c = \sum_{j=1}^{n} \sum_{k=1}^{m_i} A(T_i, k, j)S_{ij}.c
\]
\[
T_i.l = \sum_{j=1}^{n} \sum_{k=1}^{m_i} \left( \sum_{l=1}^{n} A(T_i, k, l)S_{ij}.l \right) + \omega \cdot T_i.T \cdot (T_i.l)
\]
\[
T_i.e = 1 - f(T_i, n)
\]
\[
T_i.r = \sum_{j=1}^{n} \sum_{k=1}^{m_i} A(T_i, k, j)S_{ij}.r
\]
In Formula (5), \( WT(T_i) \) refers to the waiting time and re-execution time for \( Ti \) if all the replicas for \( Ti \) fail. \( WT(T_i) \) can be predicted using average historical waiting time for \( T_i \).

B. Services selection model

The QoS-aware services selection is to select and bind a candidate service: \( CS_{ij} \) \((1\leq i\leq N, 1\leq j\leq N)\) for the replica for each \( S_i \) in each \( T_i \) in the SP (to select a composite service for a SP), as shown in Fig. 2, to maximize each user’s satisfaction degree, subjecting to multiple context constraints. We assume the user’s satisfaction degree to a composite service is determined by the global QoS value of the composite service, and one CS can only bind to one replica.

Before execution of a composite service, the global QoS calculation method is as follows:

1) Calculate the expected QoS values for all tasks in the composite service according to Eq. (4)-(7); 2) calculate the global QoS values \((Q)\) according to different structures based on the expected QoS value for each task and user-defined weights over different QoS parameters [3]; 3) normalize each global QoS value [13].

After the execution of a composite service, the global QoS value is aggregated from the QoS values of the executed replicas, and the successful execution rate of a composite service is 1 if it has been executed successfully.

In the QoS-aware service composition environment, the context information that influences the selection of candidate service [2] can be collected and transformed into a set of constraint expressions \( E \) acting on the \( j \)th \((1\leq j\leq n)\) replicas for one or more than one tasks. In that case, if a task \( T_i \) in its \( j \)th replica fails, \( T_i \) switches to its \((j+1)\)th replica, then the constrained conditions on the \((j+1)\)th replicas also need to be satisfied for all the unexecuted tasks. In resource constrained environments, the inner device resource constraints [12] should be considered: if a selected \( j \)th replica of \( Ti \) (represented by \( C(T_i) \), a selected CS) is deployed on device \( ds(C(T_i)) \), then the values of resource requirements for running \( C(T_i) \) (represented by \( ds(C(T_i)).value \)) will not exceed the current available amount of resources of \( ds(C(T_i)) \) (represented by \( ds(C(T_i)).value \)); that is, \( C(T_i).value \leq ds(C(T_i)).value \).

Moreover, to reduce the fault rate of services caused by inadequate device resources, the selected replica of the same
The faults to be handled include the service faults caused by inadequate service resources before the execution of multiple composite services, and the resource-dependent faults caused by inadequate device resources during the execution of concurrent composite services. The process of fault handling will be invoked before or during the execution of concurrent composite services for concurrent processes, each process is requested by a selfish user. We apply the auction-based multi-issue negotiation protocol [14] to handle faults for all the composite services. Our protocol places more emphasis on applying auction-based multi-issue negotiation technologies in optimizing multiple composite services for different users. The negotiation process is realized by means of conversation among multiple agents, each of which represents a participated selfish user. The result of the negotiation is to get agreed adjusted composite services or service execution strategies of composite services for all agents involved in the negotiation process. In the negotiation process, agents need to act competitively because of their self-interested nature. Each NA represents an autonomic user who requests for a composite service. When the negotiation process is invoked, each NA involved in proposes a set of updated results for multiple concurrent composite services based on the conversations among each other under the guidance and control of the CA. Since the result proposed by each NA only intend to increase each NA's satisfaction degree, the CA finds a result that maximizes the overall satisfaction degree for multiple NAs.

A. Multi-issue negotiation before composite services execute

The negotiation process occurs before execution of concurrent composite services. The aim of the negotiation process is to update the composite services for some NAs, addressing the issues of undesirable competition for service resources. The process of negotiation will be invoked when there are inadequate feasible replicas for a task (Ti) for a NA (represented by NAia). Without the negotiation process, NAia's satisfaction degree will be 0. NAia initiates the procedure of negotiation, as shown below:

**Step 1a:** NAia SENDS Ti and a "Negotiation Request" message to all NAs and CA;

**Step 2a:** each NA whose requesting SP (represented by NAia.SP) contains Ti can accept this negotiation, and NA's current composite service for execution is ComSi; NAia (1≤i≤agentNum, agentNum represents the number of agents who accept this negotiation) ACCEPTS this negotiation, then SENDS the number of Ti's and all the QoS information about Ci(Ti) (see Section III), C(Ti).Q (1≤i≤replica quantity) in NAia.SP and a "Negotiation Accept" message to CA;

**Step 3a:** if CA RECEIVES less than two "Negotiation Accept" messages, then the negotiation fails and EXITS;

**Step 4a:** CA SENDS a "Negotiation Start" message and the total number of Ti's (represented by taskNum) to each NA in response to the "Negotiation Accept" message;

**Step 5a:** each NAia MAKES a set of contract points (a bid), each of which stands for a composite service for all the taskNum×n replicas since each Ti needs at least n replicas. Each result (represented by resij, 1≤i≤resultNum, resultNum, stands for the number of contract points NAia has made, resij represents the jth contract point made by NAia) meets all the constraint expressions acting on the replicas for NAia.SP (represented by E(NAia.SP)). Using resij, NAia’s composite service can be updated to ComSUpdated by replacing the related replicas of each Ti, and NAia’s satisfaction degree to the composite service can be updated from sat(ComS) to sat(ComSUpdated);

**Step 6a:** each NAia SENDS E(NAia.SP), resij and sat(ComSUpdated) to CA;

**Step 7a:** when CA RECEIVES the information from all NAs, CA SEARCHES for the result: resij (1≤i≤agentNum, 1≤j≤resultNumi) in all the received composite services, such that resij meets \( \bigcup_{i=1}^{agentNum} E(NAia.SP) \), and \( \sum_{j=1}^{resultNumi} sat(ComSUpdated) \) is maximized, if such result does not exist, then the negotiation fails and EXITS;

**Step 8a:** CA SENDS resij to NAia (1≤i≤agentNum);

**Step 9a:** if NAia (1≤i≤agentNum) ACCEPTS resij, then CA UPDATES the composite services for the related NAs, else the negotiation fails and EXITS.

In Step 4a, the “Negotiation Start” message sent by CA to a NA means the CA will obtain the control rights for the composite service of the NA until the end of the negotiation. In Step 5a, each NAia searches contract points (results) in the nonlinear utility spaces by simulated annealing (SA) [15] or random sampling (AR), and NAia’s loss of satisfaction degree from NAia’s current composite service to NAia’s updated composite service (increase of satisfaction degree) needs to be less than a threshold \( \delta g(\delta g>0) \). \( \delta g \) is defined by NAia, it represents that the loss of the satisfaction degree NAia can tolerant. Since each NA only knows constrained conditions for its requesting SP, the CA needs to take all the constrained conditions into account.

B. Multi-issue negotiation during the execution of composite services

When multiple concurrent composite services are running in a resource-constrained environment, the aim of the negotiation process is to update the service execution strategies of the composite services for some NAs, addressing the issues of extra waiting. The process of negotiation will be invoked if the nth replica of a Ti in NAia’s requested composite service (represented by C(Ti),
n represents the replica quantity for each \( T_i \) fails due to inadequate device resources. Without the negotiation, \( \text{NA}_{ib} \) needs to wait for device resources released from other running composite services. \( \text{NA}_{ib} \) initiates the procedure of negotiation, as shown below:

**Step 1b:** \( \text{NA}_{ib} \) SENDS \( T_{ib}, C_i(T_i), C_{i}(T_i).value, ds(C_{ib}(T_{ib})), ds(C_{ib}(T_i)).value, \) and \( C_i(T_i).Q \) to all the NAs and CA;

**Step 2b:** each NA SEARCHES for its currently running replica of the task \( T_j; C_i(T_j), \) if \( ds(C_i(T_j)) \) equals \( ds(C_{ib}(T_{ib})) \), then the NA can accept this negotiation; \( \text{NA}_{ib} \) (\( 1 \leq \text{agentNum}' \), \( \text{agentNum}' \) represents the number of agents who accept this negotiation) ACCEPTS this negotiation, then SENDS \( T_{ib}, C_i(T_i).Q, ds(C_i(T_j)), C_i(T_j).value, ds(C_{ib}(T_{ib})).value (1 \leq i \leq n) \) and “Negotiation Accept” message to CA;

**Step 3b:** if CA RECEIVES less than two “Negotiation Accept” messages, then the negotiation fails and EXITS;

**Step 4b:** CA TRANSFORMS some of the received information into a set of constraint expressions about device resource constraints (represented by \( E(\text{device}) \)), then SENDS a “Negotiation Start” message, all the received information and \( E(\text{device}) \) to each NA in response to the “Negotiation Accept” message;

**Step 5b:** each \( \text{NA}_{ib} \) MAKES a set of contract points (a bid), each of which stands for a set of switches for each received task from the current replica to the next replica. Each composite service can meet \( E(\text{device}) \) and does not need to wait for the current task if each set of the switches happens. For one contract point, \( \text{NA}_{ib} \) CALCULATES the satisfaction degree to each candidate service execution strategy and the satisfaction degree to each updated service execution strategy if each set of switches happens. In one set of switches, if \( \text{NA}_{ib} \)’s currently executing replica for a task \( T_j \) has changed from the current \( u^b \) replica to the \( v^b \) replica, \( \text{NA}_{ib} \)’s satisfaction degree may be reduced if \( u^b \) \( \neq v^b \) and \( C_i(T_j) \) is currently running, since \( C_i(T_j) \) has to be canceled; and \( \text{NA}_{ib} \)’s satisfaction degree may be increased if \( C_i(T_j) \) is currently blocked due to inadequate device resources, since \( C_i(T_j) \) needs not to wait;

**Step 6b:** each \( \text{NA}_{ib} \) SENDS the bid and all the calculated information to CA;

**Step 7b:** when CA RECEIVES the information from all NAs, CA SEARCHES for a contract point shared among all bids, such that the overall satisfaction degree to all the service execution strategies after the related set of switches is maximized and larger than the degree before negotiation, if such result does not exist, then the negotiation fails and EXITS;

**Step 8b:** CA SENDS the contract point to \( \text{NA}_{ib} \) (\( 1 \leq \text{agentNum}' \));

**Step 9b:** if \( \text{NA}_{ib} \) (\( 1 \leq \text{agentNum}' \)) ACCEPTS the contract point, then CA UPDATES the service execution strategy for the related NA, else the negotiation fails and EXITS.

In Step 2b, the CA checks the trustworthy of the submitted information by collecting the information from different composite services periodically, and dishonest NAs will be denied for the negotiation procedure. In Step 5b, for each \( \text{NA}_{ib} \), its each contract point, its corresponding loss of satisfaction degree (increase of satisfaction degree) should be lower than a threshold \( \delta_{ib} \). \( \delta_{ib} \) is defined by \( \text{NA}_{ib} \), and the waiting time can be calculated according to Formula (5) for \( \text{NA}_{ib} \) whose current replica is blocked due to inadequate device resources based on the received information.

In reality, the NA who initializes the procedure of negotiation needs to pay a fee to other NAs who accept the negotiation procedures for their loss of satisfaction degrees if necessary. In Step 9b, all of NAs involved in the negotiation procedure can accept the contract point since it is shared among all the bids.

V. ADAPTIVE CONTEXT-AWARE SERVICE RE-SELECTION

In order to optimize each composite service at runtime continuously, each context manager monitors the dynamic context of one composite service by collecting, predicting and transforming the context information periodically. The transformed context information includes all QoS values of all candidate services of unexecuted tasks, constraint expressions acting on the replicas of all unexecuted tasks, etc. Then each context manager determines whether to re-select replicas for all unexecuted tasks in the composite service based on an adaptive context-aware service re-selection mechanism.

A. Resource-dependent fault rate \( (f_b) \) prediction

In our framework, we assume all QoS values (expect the \( f_b \) and successful execution rate \( (e) \)) of each candidate service can be collected directly from distributed service providers. Since the \( f_b \) of a service influences its successful execution rate, we focus on predicting the \( f_b \) of each candidate service.

In the context-aware service re-selection mechanism, the context manager needs to predict the \( f_b \) of all the candidate services periodically for all unexecuted tasks based on the transformed context information when the composite service is running. The resource-dependent faults will occur when there are inadequate device resources for multiple concurrent services at the same time on the same device. Therefore, for one service \( s \), its resource-dependent fault rate (denoted by \( f_b(s) \)) will be higher as its device resource utilization (denoted by \( s.value/ds(s).VALUE \), where \( VALUE \) represents the amount of all resources in a device) increases.

In addition to the context manager, each NA also participates in the process of \( f_b \) prediction. Once the \( n^b \) replica of a task \( T_b \) fails during the execution due to inadequate device resources when the composite service is running, the NA may receive the information about the failed replica \( C_{ib}(T_b) \) including \( C_{ib}(T_b).value, ds(C_{ib}(T_b)) \), \( ds(C_{ib}(T_b)).value, \) etc., where, if \( ds(C_{ib}(T_b)) \) equals \( ds(s) \), the predicted \( f_b(s) \) will be updated using Eq. (10):

\[
f_b(s) = f_b(s) + \beta \frac{C_{ib}(T_b).value}{ds(C_{ib}(T_b)).value}
\]

predicted \( f_b(s) \) is \( \alpha.s.value/ds(s).VALUE \), \( \alpha \) and \( \beta \) are parameters defined by each user, and \( 0<\alpha, \beta<1 \).
B. Context-aware service re-selection

Since all context information can be collected or predicted by the context manager, and we discovered all the changing contexts can be reflected in the following conditions, thus, we set the following conditions for service re-selection for all the unexecuted subsequent tasks in each composite service:

**Condition 1:** One of the bound replicas which may be executed for the subsequent tasks has been removed, or one of the constraint expressions acting on the replicas of the subsequent tasks cannot be met;

**Condition 2:** The QoS values of a number of (denoted by QBNum) bound replicas for the subsequent tasks have been changed;

**Condition 3:** A number of (denoted by conNum) constraint expressions acting on the unbound CSs for the subsequent tasks have been changed;

**Condition 4:** The QoS values of a number of (denoted by QUBNum) unbound CSs for the subsequent tasks have been changed;

**Condition 5:** A number of (denoted by CNum) unbound CSs for the subsequent tasks have been removed or added;

**Condition 6:** The composite service has determined its choice or execution times at the “choice” or “loop” structures.

In Condition 3, the constraint expressions which contain the element of the current available amount of resources of a device are not taken into account, since they may always change. In Conditions 2 and 4, the \( f_b \) of a CS is regarded to be changed if its predicted value is higher than a threshold \( \delta_c \). Once a service re-selection process is invoked due to Condition 2 or 4, the predicted values of \( f_b \) of all CSs will return to the initial predicted ones. A service re-selection process will be invoked when Condition 1 is met. In addition, the strategy adaption module will analyze the benefit and cost while selecting some of these conditions for re-selection based on users’ historical execution strategies, and determine an optimal combination of conditions which can optimize the user’s service satisfaction degree while meeting the user’s computational cost constraint.

VI. IMPLEMENTATION AND EXPERIMENTS

To study the performance of our approach, we used an Internet topology generator Inet 3.0 [17] to create a number of nodes for presenting different candidate services (CSs) in the Internet. Then different CSs were selected to generate the SP described in Fig. 2 for each user. All the service contexts, device contexts, and other QoS related contexts were simulated and stored in XML files and could be updated automatically by a context generator. In order to realize the framework in Fig. 1, we used NetLogo [18] to simulate complex interactions among the NAs, CA and the context managers. Different experiments were conducted in different environments with different device numbers (devNums), NA numbers (NANums) or CS numbers for each task (CSNums). The performance metrics of composite services analyzed in the experiments include the averages (AVG) and standard deviations (STDEV) of the satisfaction degree (Sat) and waiting time (WT) of the NAs. Compared with the service execution time for each CS, the time spent in running the repair genetic algorithm and the time spent in the negotiation procedures could be neglected in our experiments. The ranges of QoS values of \( c, l, f_a, \) and \( r \) (see Section III) were [0, 1], [100, 200], [0.1, 0.3], and [0, 1], respectively, and there was one constraint [2] in other QoS related contexts. We fixed \( \alpha \) to 1, \( \beta \) to 0.2, and threshold \( \delta_c \) to 0.8. All results of our experiments were obtained from six different test cases.

A. Advantages of the negotiation process before composite services execute

To analyze the benefits of the negotiation process before the execution of multiple concurrent composite services, we constructed several NAs requesting composite services and a CA, limited CSs for each task, and simulated the negotiation procedure among them from step 1a-9a. Before composite services execute, each NA required at least two replicas for each task in the same SP while each NA could occupy four replicas for each task at most. We compared the performances of all composite services in the same environment with and without the negotiation process (Neg and nonNeg) in each test case. The number of exclusive constraints [10] in other QoS related contexts was 20.

As depicted in Fig. 3, the negotiation process can help to increase the average satisfaction degree of multiple NAs and make satisfaction degrees among multiple NAs more even distributed in the environments with limited service resources for SPs. The advantage of the negotiation process will get clearer when the service resources are getting more limited for all NAs (i.e. when the CS number is less or the NA number is more). However, if the service resources are...
too limited, the advantages will become less obvious since the negotiation will be more likely to fail.

We also set different values for the threshold $\delta_{ia}$ ($1\leq i\leq agentNum$) and analyzed the performances with and without the negotiation process in Fig. 3. We can see that our negotiation process has more effects on increasing the average satisfaction degree of multiple NAs and making the satisfaction degrees among multiple NAs more even distributed as the values of $\delta_{ia}$ decrease, since the successful negotiation rate will increase.

However, when $\delta_{ia}$ is very low, each NA has to submit almost all bids (composite services) to the CA, then, the CA searches for all possible combinations of the submitted bids that maximize the sum of the satisfaction degrees, therefore, the data transmission cost and computational cost will increase. Thus, the completeness and the computational cost are a trade-off relation in the negotiation process. In practical systems, $\delta_{ia}$ can be determined and adjusted by each user according to different requirements, which makes the mechanism more adaptive.

**B. Advantages of the negotiation process during the execution of composite services**

In order to analyze the advantages of the negotiation process during the execution of concurrent composite services, we simulated the device contexts with limited device resources for running multiple composite services for multiple SPs concurrently, where the amount of all the resources in a device (VALUE) ranged: [60, 70], the value of resource requirements for running a CS ranged: [20, 30]. Each NA required three replicas for each task in the same SP. We simulated multiple concurrent running SPs in the same environment with limited device resources, and compared the performance of all the composite services with our negotiation mechanism when there were inadequate device resources, and the performance with the first come first service (FCFS) mechanism in the same environment in each test case. In the FCFS mechanism, the replica which arrives early in a SP occupies the required device resources for running while other replicas may need to be waiting for the released device resources, and there is no communication among NAs.

Fig. 4 shows the effects of our negotiation process during the execution of concurrent composite services by increasing the average satisfaction degree, decreasing the average waiting time of multiple NAs, and averaging the satisfaction degrees and the waiting time among multiple NAs in the environments with limited device resources for the SPs. The advantage of the negotiation mechanism is more obvious when the device resources are getting more limited but not too limited. Likewise, the negotiation threshold $\delta_{ib}$ has similar effects in influencing performances of multiple concurrent composite services, data transmission cost and computational cost as the threshold $\delta_{ia}$ in the experiment above.

![Figure 4. Performances comparison after execution of multiple concurrent composite services with the negotiation process in execution and without the negotiation process (CSNum: 25).](image)

**C. Performance analysis of the adaptive context-aware service re-selection mechanism**

The strategy adaptation module records the service execution information about each historical composite service for each NA in a dynamic environment. The execution information includes the selected combination of conditions and got several related service execution records for NA$_1$, as shown in Table 1.

We set a new combination of conditions for service re-selection for NA$_1$, ran the service re-selection mechanism, and let the composite service ran six times in the same dynamic environment. Then, we recorded the service execution information. We have set several different combinations of conditions and got several related service execution records for NA$_1$, as shown in Table 2.

When a new similar composite service comes for execution for NA$_1$ in the similar dynamic environment, and
NA1 can afford 80 times of re-selection for the composite service, the mechanism will select conditions 1, 2, 3, 4, 5 as the best combination.

**Table 1** Parameters in the dynamic environment

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NANum</td>
<td>2</td>
</tr>
<tr>
<td>devNum</td>
<td>2</td>
</tr>
<tr>
<td>initial CSNum</td>
<td>25</td>
</tr>
<tr>
<td>number of replicas for each task</td>
<td>2</td>
</tr>
<tr>
<td>initial number of exclusive constraints</td>
<td>10</td>
</tr>
<tr>
<td>cycle of adding or removing a CS</td>
<td>2s</td>
</tr>
<tr>
<td>cycle of changing an exclusive constraint</td>
<td>10s</td>
</tr>
<tr>
<td>cycle of changing a QoS value of a CS</td>
<td>2s</td>
</tr>
<tr>
<td>QNum</td>
<td>7</td>
</tr>
<tr>
<td>conNum</td>
<td>4</td>
</tr>
<tr>
<td>QUBNum</td>
<td>80</td>
</tr>
<tr>
<td>CNum</td>
<td>100</td>
</tr>
</tbody>
</table>

**Table 2** Combinations of conditions and related service execution records

<table>
<thead>
<tr>
<th>Combination of conditions for re-selection</th>
<th>AVG Sat</th>
<th>AVG Re-selection times</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8904</td>
<td>4</td>
</tr>
<tr>
<td>1, 2</td>
<td>0.8953</td>
<td>20</td>
</tr>
<tr>
<td>1, 2, 3</td>
<td>0.9013</td>
<td>29</td>
</tr>
<tr>
<td>1, 2, 3, 4</td>
<td>0.9114</td>
<td>63</td>
</tr>
<tr>
<td>1, 2, 3, 4, 5</td>
<td>0.9195</td>
<td>78</td>
</tr>
<tr>
<td>1, 2, 3, 4, 5, 6</td>
<td>0.9207</td>
<td>82</td>
</tr>
<tr>
<td>1, 2, 5</td>
<td>0.9035</td>
<td>37</td>
</tr>
<tr>
<td>1, 2, 6</td>
<td>0.8976</td>
<td>23</td>
</tr>
</tbody>
</table>

**VII. RELATED WORK**

For QoS-aware service composition in dynamic environments, many proposed methods adopt service backup mechanisms to increase the availability of composite services: when one concrete service fails or its QoS values degrade dramatically, the composite service can easily switch to a replacement, to avoid an extra delay [4]. Other researchers utilized service negotiation [5] to improve the performance of QoS-aware service composition: if a feasible service execution solution cannot be obtained, a service negotiation process will be invoked between a service requester and multiple service providers, to reduce composite service invocation failures. Other adaptive approaches include triggering the service re-selection process for unexecuted tasks when the composite service is running and the actual QoS values of the concrete services deviate from the initial estimations [6], or the context similarity between the current context information and the historical one for the current execution plan is below a threshold [2]. Performance prediction [26] was also incorporated into the adaption of composite services, which enables the composite service to heal by itself from a failure as quickly as possible and minimizes the number of re-plans.

In our approach, we extract six conditions reflecting the changing contexts when each composite service is running, and help each user to analyze the benefit and cost for selecting a collection of conditions for service re-selection based on the historical records, which makes the re-selection mechanism more user-customized.

In QoS-aware service compositions, the context issues, including bandwidth [19], policy context [10], device context [12], response time constraints [23], etc, were taken into account. Usually, such problems were modeled as constraint optimizing problems and solved by linear programming [1, 5, 19], genetic algorithms [2, 9] or hybrid approach [25]. In our approach, we use the repair genetic algorithm for solving the service composition problem before execution of concurrent processes, since GA performs well in effectiveness, efficiency, and adaptability in a dynamic environment with multiple and variable constraints. Currently, some new context issues including communication latencies [24] between services during the composition phase was also taken into account. Our model can take all the dynamic local and global constraints for each process into account for service composition, and focus on the device constraints among multiple processes.

Agent technologies were also incorporated into the composition. The composition agent [20] was proposed to support composition based on semantic information and QoS properties, which enables a requester to work smoothly with fewer loads. Other researchers proposed the multi-agent based QoS-aware service composition solution (MQSC) [21], to ensure the end-to-end QoS of the composite service.

Considering many concurrent tasks for composition in resource limited environments, in [22], a mathematical model using game theory to depict the competitive relationship between multitasks and Web service under QoS constraints was established, addressing the issues of undesirable competition for service resources. Compared with this model, our approach focused on re-allocating service and device resources before and during the execution of whole concurrent processes for different users, and each user can determine proper level of participation to the negotiation process for resource re-allocations by setting different parameters.

**VIII. CONCLUSIONS**

This paper aims to optimize the QoS-aware services composition for multiple concurrent processes for multiple selfish users. In order to solve the issues of undesirable competition for service resources and extra waiting, we used a multi-issue negotiation protocol among agents based on an auction-based multi-issue negotiation protocol to re-allocate the service resources and device resources before and during the execution of concurrent composite services. In addition, we proposed a context-aware service re-selection mechanism to make each composite service more adaptive to the dynamic context environment. Experiment results have demonstrated the advantages of our methods in a dynamic resource-constrained environment. In future we will analyze the costs in communications among multiple
agents during the negotiation processes, and compare our negotiation protocols to other related works. In addition, we will try to incorporate our model into enterprise service scenarios.

REFERENCES


