SOCA (2012) 6:15–32
DOI 10.1007/s11761-011-0096-5

SPECIAL ISSUE PAPER

A sharing-oriented service selection and scheduling approach for the optimization of resource utilization

Zhongjie Wang · Xiaofei Xu

Received: 26 March 2011 / Revised: 24 November 2011 / Accepted: 6 December 2011 / Published online: 25 December 2011

Abstract We present a sharing-oriented service selection and scheduling approach capable of finding a trade-off between requirement satisfaction degree, service utilization rate and service sharing cost for limited quantities and capacities of available services. In traditional service selection approaches, each customer requirement is independently satisfied by optimally selecting a set of candidate service resources. However, in real-life service scenarios, it is usual for multiple customers to raise their requirements simultaneously, and available services need to be allocated between them. Especially, when available services are limited in both quantity and capacity, a traditional "first-come-first-serve" strategy would lead to a low service utilization rate, and some requirements cannot be satisfied at all (i.e., a low requirement satisfaction degree). Our approach makes use of the feature that some services can be shared by several customer requirements. Specifically, a virtualized service resource consisting of multiple candidate services is constructed and scheduled to satisfy multiple customer requirements simultaneously. Our approach searches for the global optimization on requirement satisfaction degree, service utilization rate, and service sharing cost. We build a mathematical model for this multi-objective optimization problem and propose a nested genetic algorithm mixed with a greedy strategy. Experiments in an ocean transportation service setting are conducted and our approach is compared with traditional approaches to validate its effectiveness.

Z. Wang (✉) · X. Xu
Research Center of Intelligent Computing for Enterprises and Services (ICES), School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China
e-mail: rainy@hit.edu.cn
URL: http://www.hit.edu.cn

X. Xu
e-mail: xiaofei@hit.edu.cn

Keywords Service selection · Service sharing · Requirement bundling · Virtualized service resource · Service utilization rate

1 Introduction

With the development and prosperity of the IT-enabled service industry (for example, e-Business, production services, and the "Internet of Things"), there has been an increasing number of e-services in the form of web services, business processes, information portals, and so on [1,2]. In e-service, real-world services are virtualized and provided online, and providers and customers then collaborate through a combination of online and real-life interactions to complete the service process. Such a process is composed of a set of behaviors, and various service resources are used to fulfill these behaviors.

From business model's point of view, in the face of the growing number of customers and e-service providers, bilateral resource integration services (BIRIS) [3] have become one of the predominant patterns in today's service industry. An emphasis of BIRIS is that the demand-supply relationship is established through an intermediary broker between customers and providers, and the complex requirements raised by customers are to be satisfied by a set of distributed service resources provided by multiple providers [4]. Typical examples include Alibaba.com (a B2B broker), Taobao.com (a C2C broker), and Ctrip.com (a tourist service broker).

In BIRIS-oriented services, a key feature referred to as *multi-requirement* (or the *requirement queue*) is the massive requirements continually raised by multiple customers. The intermediary broker faces the important issue of how to fully use the currently available resources to ensure that these demands are fulfilled as much as possible, while also ensuring the maximum use of resources so as to reduce both idleness and waste. In summary, resources provided by different enterprises need to be dynamically bundled and configured on demand to satisfy a set of complex needs [4].

Taking tourism services as an example, multiple customers make travel requests (including flight and hotel bookings, tourist bus bookings, tourist attraction ticket bookings, and so on). To consider each individual demand is unrealistic (for example, it is impossible to arrange a bus for one customer) but multiple demands should be taken into consideration simultaneously, thus efficiently using the service resources (that is, a bus is allocated to 20 customers). The same scenario occurs in logistics services. Suppose one customer requests that half a container of goods be transported and another that one and a half containers of goods be transported. If the demands are considered separately, three containers are required, but if they are bundled, then two containers are sufficient [5].

This scenario abounds in real-world services and is of interest not only to a single provider who owns limited resources but also to intermediaries who match customer requirements (CRs) to what is offered by various providers in their catalogs [6].

From the point of view of research, this problem can be considered in one of three ways:

1. A service selection and composition problem [7], that is, service components (usually behavior-oriented functions) are selected and composed together to satisfy the requirements;
2. A service bundling problem [8], that is, multiple service products (resources or behaviors) are bundled and holistically provided to customers; or
3. A scheduling problem [9], that is, service resources are allocated to multiple requirements with the specified quantities and scheduling.

However, most service composition and service bundling approaches only consider a single demand raised by a specific customer or aggregated from a group of similar customers. Thus, the "sharable" feature of services is not considered and the service utilization rate (SUR) is not regarded as an optimization objective.

As a consequence, in the face of multiple requirements, the one-requirement-oriented service composition/bundling method has to be repeatedly applied for each requirement. This leads to a situation where the resource utilization rate remains low and the continuity of CR satisfaction is poor. This approach is shown schematically in Fig. 1, where each $cr_i (i = 1, 2, \ldots, n)$ is a CR and each $sr_j (j = 1, 2, \ldots, m)$ is an available service resource. If sr_1 and sr_2 are selected to satisfy cr_1, they will not be considered for other requirements any longer, even though cr_1 might not occupy their full capacity.

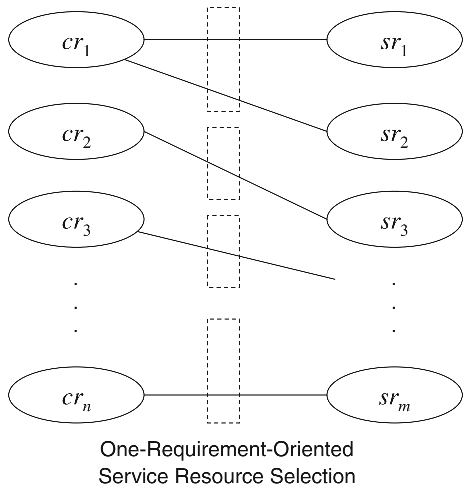

Fig. 1 Traditional service composition approach for satisfying multiple requirements

If we observe that there are similarities between many requirements and that some resources can be used in a shared manner between multiple requirements, then we could consider all requirements simultaneously when allocating/scheduling resources to them, thus improving both the extent to which the requirements are satisfied and the utilization rate of the available resources.

Another issue is that when considering the constituent elements of service bundling/composition, the current research concerns mainly those services that are accessed completely through the Internet without any human participation, such as Internet connection services, telephone services, information/database services, and web services. The composition of such services is comparatively easy because QoS optimization is the principal objective and the only sharing mechanism between them is "concurrency". However, in real-world services, there are many resources that cannot be simultaneously accessed, examples being physical resources and human resources. These resources can be virtualized and publicized on the Internet in the form of WS-HumanTask, WS-Resource, and so on, but their use and execution are still in the real world. The scheduling and use of these resources differs from traditional e-services because of the time- and space-sharing characteristics that have seldom been considered before.

This paper considers the following scenario. For multiple demands $CR = \{cr_1, cr_2, \ldots, cr_n\}$ raised by multiple customers and a set of available service resources $SR = \{sr_1, sr_2, \ldots, sr_m\}$, we construct a virtualized service resource (VSR) and a corresponding resource scheduling scheme to allocate *m* resources in *SR* among *n* requirements in *CR*. The optimization objectives include (1) maximizing

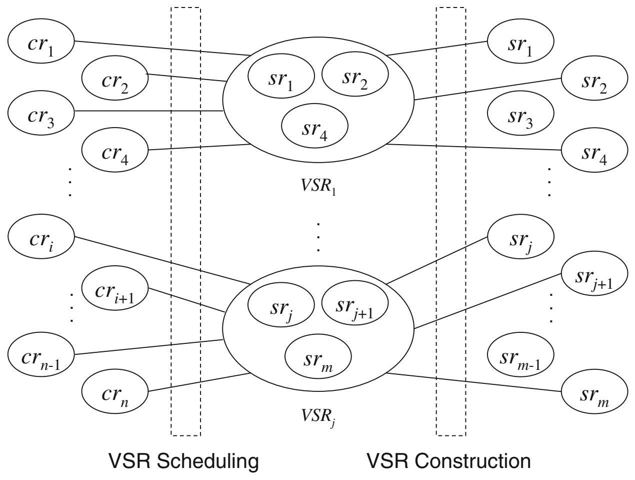

Fig. 2 Our VSR-based service composition approach for satisfying multiple requirements

the number of satisfied customer requirements, (2) maximizing the utilization rate of resources, and (3) minimizing the cost incurred by the resource sharing. The approach is depicted in Fig. 2. For example, VSR_1 consists of three resources $\{sr_1, sr_2, sr_4\}$ and they jointly satisfy four requirements $\{cr_1, cr_2, cr_3, cr_4\}$.

This paper is organized as follows. Section 2 summarizes related research. Section 3 gives the necessary preliminaries, including the descriptions of customer requirements (CR), service resources (SR), and the virtualized service resource (VSR). Section 4 evaluates VSR and its scheduling using three metrics. Section 5 presents the VSR construction and scheduling methods. Experiments and comparisons are conducted in Sect. 6. The conclusions and suggestions for future work are presented in Sect. 7.

2 Related work

Multi-requirement-oriented resource selection and scheduling is actually a combination of service selection/composition, service bundling, and service resource scheduling. This section briefly reviews the current approaches in these three areas.

Service selection/composition is a topic of interest in services computing research, and numerous methods have been presented in the literature (see surveys such as [7,10,11]). The dominant approach is semantics-based dynamic composition. That is, customer requirements and candidate service resources are described by formal modeling languages, and various semantics-matching techniques or artificial-intelligence-based planning techniques are adopted to search for an optimized composition solution. The current research aims are: (1) to facilitate the transformation from manual composition to full-automatic composition by adopting semantics reasoning and artificial intelligence so as to improve efficiency and accuracy; and (2) to facilitate the transformation from static composition to dynamic composition and realize dynamic adaptation by adopting context and QoS-awareness so as to improve agility.

Note that only a few papers concern the "multi-requirement" scenario. Cardellini et al. [12] presented a method in which multiple requirements arriving at the same moment or in the same time interval are grouped together and handled in batch processing. However, the objective of this approach is to improve the efficiency instead of maximizing the resource utilization rate. Wang et al. [13,14] proposed an equilibrium-oriented service composition approach with the objective of preserving the equilibrium between the satisfaction degrees of multiple demands with limited available services. Again, the service sharing feature is not considered.

There have been many research outcomes from service bundling studies. Research first began with product bundling and extended to service bundling (for example, telecommunication services and software services), and then further to product-service mixed bundling. Service bundling has received wide attention regarding price discrimination, increasing sales, promoting customer lock-in and creating entry barriers [15].

A service bundle, also called a service module [16] or complex service [17], is a package of independent elementary products or services offered by different decentralized providers [4,8]. These elementary services are combined in a network topology that is shaped by service configurations, interrelations, and dependencies [17]. Research on service bundling has included work on the principles and objectives of service bundling [18], the description and modeling of service bundles [15,19], including typical relationships between multiple elementary services, pricing of the service bundle [17,18], methods and processes of service bundling [4,18–20], and quality evaluation of service bundling [8].

Resource allocation and scheduling is a very classical problem in many areas of research, such as operation research, grid computing, and cloud computing. Concerning the job-shop scheduling in manufacturing domain where a finite set of jobs is to be allocated to specific time intervals on a finite set of machines (equipment resources), optimization techniques like genetic algorithms, tabu search, reinforcement learning, artificial neural networks (see survey in [21]) are employed to find a schedule of minimum length.

In grid computing and cloud computing, resource scheduling looks for a set of computing resources (CPU, storage, software, etc) for each client's request and ensures an appropriate level of QoS [22], with the help of protocols like the Globus Resource Allocation Manager (GRAM) and schedulers like PBS (Portable Batch System). GRAM allows users to run jobs remotely, providing an API for submitting, monitoring, and terminating the jobs [23]. PBS is a batch job and computer resource management and scheduler consisting of one Job Server, one or more Job Schedulers, and one or more execution servers [24]. It accepts batch jobs, allocate the job to the proper computing resources, preserve and protect the job until it is run, run the job, and deliver output back to the submitter. Each physical resource is defined as a "node", and the node consisting of one or more virtual processors are defined as a "cluster node". A set of nodes could be assigned exclusively to a job for the duration of that job (such nodes are called "exclusive node"), or their virtual processors are temporarily shared by multiple jobs, i.e., to allow multiple jobs to run concurrently on one node (called "temporarily shared nodes") [25]. PBS job scheduler adopts load balance policy to look for a distribution of jobs across multiple exclusive and time-shared nodes to pursue optimized performance, throughput, and scalability.

Compared with GRAM and PBS, our scenario looks wider, i.e., it does not consider pure computing resources but transfers to real-life physical and software resources. Besides the exclusive and time-sharing scheduling strategies, we consider other new sharing mechanisms (e.g., concurrent sharing and space sharing). In addition, the resource availability and QoS are more dynamic.

3 Preliminaries

3.1 Service

We assume that all customers request the same service S which requires p types of service resources $\{T_1, T_2, \ldots, T_p\}$. For example, ocean transportation services (OTS) include four primary types of resources, namely *ship cabins*, *vehicles*, *containers*, and *customs declaration e-services*.

3.2 Customer requirements

Without loss of generality, we define a customer requirement (cr) to be

$$cr = (C, RR, RT, EST, LST)$$

where

1. C is the customer;
2. $RR = \{rr_i\}$ is a set of detailed requirements on the concrete service resources: $rr_i = \langle T,\ Q^R,\ P,\ QoS,\ LST,\ D\rangle$ where
   - T: the requested resource type;
   - Q^R: requested quantity;
   - $P = [LP,\ HP]$: the expected price range with lowest and highest price;
   - $QoS = [LQ,\ HQ]$: the expected range of integrated quality metrics aggregated from a set of expected QoS parameters (availability, reliability, and so on);
   - LST: the latest start time for using the required resources; and
   - D: the duration for which the resources are required;
3. RT is the time that cr was submitted; and
4. EST and LST are the earliest and latest time when cr should be satisfied.

An OTS example might be a consigner C_{01} who requests four types of resources, container (rr_1), vehicle (rr_2), ship cabin (rr_3), and customs service (rr_4), each with specific quantities, price ranges, and QoS ranges. The requirement is submitted at 3:00 pm, January 30, and transportation of the goods is expected to start on or after January 31 but not later than February 2.

3.3 Requirement segmentation

Multiple requirements raised in a specific period of time form a requirement queue $CQ = (cr_1,\ cr_2,\ \ldots)$. Because all requirements have a specified time frame, it is necessary to partition the queue into segments, each of which is to be processed simultaneously.

We present here a simple partitioning method:

1. at time t, the first requirement that has not yet been dealt with in the queue, cr_0, is added to current segment;
2. each cr_i in the queue is checked; if $cr_i.EST \le cr_0.LST$, then add cr_i to current segment; otherwise, cr_i will be considered in the next segment.

The issue of how to preserve the equilibrium between the current segment and the upcoming requirements (that is, preserving limited resources for future requirements) is considered in detail in [13]. In the remainder of this paper, we investigate the service selection and scheduling issues for a single segment in a specific time scope.

3.4 Candidate service resources

A service resource is generally defined as a virtual or physical entity that can provide specific capacities and behaviors. Products, equipment, environment, people, software, and information are all typical service resources.

Service resources are categorized according to the behaviors that they can deliver (the attribute T mentioned in Sect. 3.2). To satisfy a customer's requirements (for example, two 40-feet containers are requested), concrete resources are to be selected from the available resource pool (two 40-feet containers—No. 0232A and 0237K—are selected).

A concrete service resource is defined by $sr = (ID, Provider, T, ShareType, P, QoS, MC, MAP, MSU)$, where *ID*, *Provider*, and *T* are the unique identity, the owner, and the type of *sr*, respectively; *ShareType* is the manner in which *sr* is to be shared by multiple requirements; *P* is the unit price; *QoS* is the value of the integrated quality metrics attached to *sr*; and *MC, MAP*, and *MSU* are sharing-related attributes that will be discussed in the next section.

3.5 Sharing types for service resources

The four types of resource sharing we consider are concurrent sharing (CS), time sharing (TS), space sharing (SS), and exclusive sharing (ES). Detailed information about these and their related attributes is shown in Table 1.

3.6 Virtualized service resource (VSR)

In our approach, a number of customer requirements *CR* can be jointly completed by a number of service resources from multiple providers. We introduce a new term, virtualized service resource (VSR), to describe this *many-to-many* relationship between *CR* and *SR*.

A VSR does not exist permanently, and only when new requirements are raised does it come into being according to specified criteria. After the requirements are fulfilled, it is dismissed. VSR is quite similar to "virtual enterprise" in the manufacturing industry [26].

A VSR is defined by $VSR = (CR, SR, Scheduling)$, where CR is the set of requirements, *SR* is the set of concrete resources, and for arbitrary $cr_i \in CR$, cr_i is satisfied by the scheduling of resources in *SR*. The scheduling information is described by *Scheduling*, which includes whether and when each sr_j is to be used for cr_i, the quantity of sr_j to be allocated to cr_i, and the duration for which cr_i will occupy sr_j. *Scheduling* is detailed in the next section.

3.7 Scheduling of virtualized service resource

We write Scheduling $(VSR, CR) = \{AS\}$ to denote the scheduling schema composed of a set of atomic schedules $AS = (rr, sr, Q^S, ST, D)$. This means that for *rr*, we allocate the resource *sr* with quantity Q^S from the time point *ST* for duration D. $Q^S = 1$ indicates that *sr* is allocated entirely to *rr*, and $0 < Q^S < 1$ indicates the proportion of *sr* is allocated to *rr*.

This scheduling schema can be presented graphically as a resource allocation diagram. Figure 3 shows a schematic example, in which sr_1 will be shared by rr_1 to rr_5, sr_2 is to be shared by rr_6 to rr_8, and sr_n is used exclusively by rr_9.

When resources in *VSR* are scheduled, it is necessary to consider the sharing type for each resource. There are four categories of scheduling:

- Concurrency-based scheduling

For a concurrent-sharing resource sr_j, multiple requirements are allowed to use sr_j at the same time, but the number of

Table 1 Classification of resource sharing types

Sharing type	Descriptions	Examples	Parameters
Concurrent sharing (CS)	A resource can be simultaneously accessed by multiple customers and there is no interplay between them. The concurrency number should not exceed the maximal concurrency number (MC)	Software resources (e.g., web services), Information resources	Maximal concurrency number (MC)—MC is defined as a positive integer. If MC = 1, this resource is degenerated into an exclusive sharing one
Time sharing (TS)	A resource can be used by multiple customers in different periods in a lockstep style during the maximal available period (MAP)	Human resources, physical equipment (e.g., vehicles)	Maximal available period (MAP)—MAP is defined as a time interval with an earliest and a latest time. Outside the interval, this resource cannot be allocated
Space sharing (SS)	A resource provides space capacity that is to be shared by multiple customers. Each customer occupies part of space that should be any multiple of the minimal sharing unit (MSU)	Containers, packages, tour groups	Minimal sharing unit (MSU)—MSU is a decimal, e.g., MSU = 0.2 means that the minimal space allocated to one requirement is 20% of the whole available space
Exclusive sharing (ES)	A resource cannot be shared at any dimension but is to be used in an exclusive way, that is, it can serve only one single requirement during the whole service process	Ship cabins, tangible products	N/A

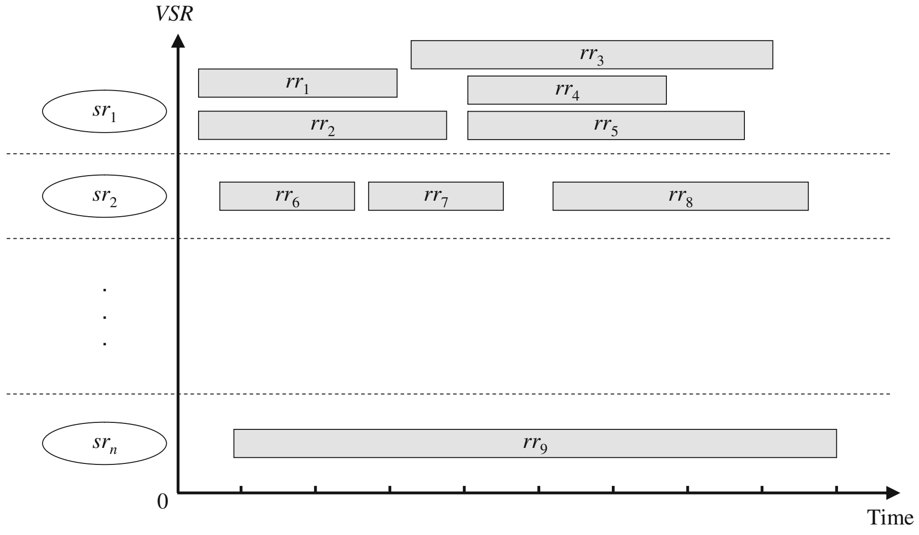

Fig. 3 VSR scheduling

requirements with simultaneous access at any time should not exceed the maximal concurrency number. That is, at all times t, $|\{AS\,|AS \in W \wedge t \in [AS.ST, AS.ST + AS.D]\}| \leq sr_j.MC$, where $W = \{AS|AS \in Scheduling \wedge AS.sr = sr_j\}$.

- Space-based scheduling

For a space-sharing resource sr_j, sr_j is allocated to multiple requirements, each of which occupies a part of the available space. The total allocated space should not exceed the maximum space of sr_j, and each requirement should be allocated a multiple of the minimal sharing unit (MSU) of sr_j to. That is, $\sum_{AS\in W} AS.Q^S \leq 1$ and $AS.Q^S = K \times sr_j.MSU$ (K is an integer). For instance, if a container "0232A" (with $MSU = 0.2$) is shared by two consigners' requirements (rr_1 and rr_2) with quantities 0.4 and 0.42, respectively, then there are two scheduling: $< rr_1$, "0232A", 0.4, $ST,\ D >$ and $< rr_2$, "0232A", 0.6, $ST,\ D >$.

- Time-based scheduling

For a time-sharing resource sr_j, the available time for sr_j is allocated to multiple requirements with no overlaps. That is, for $AS_1,\ AS_2 \in W$, if $AS_1.sr = AS_2.sr$, then $[AS_1.ST,\ AS_1.ST + AS_1.D] \cap [AS_2.ST,\ AS_2.ST + AS_2.D] = \emptyset$. For instance, if vehicle H53172 is arranged to pick up goods from two consigners (rr_1 and rr_2), then there are two scheduling: $< rr_1$, "H53172", 1, 15:00, 30min $>$ and $< rr_2$, "H53172", 1, 15:30, 72min $>$ meaning that the vehicle serves rr_1 from 15:00 to 15:30 and then serves rr_2 from 15:30 to 16:42.

- Exclusion-based scheduling

For an exclusive-sharing resource sr_j, sr_j is allocated to a single requirement without sharing. That is, for arbitrary $AS_1,\ AS_2 \in W,\ AS_1.rr \neq AS_2.rr$.

4 Evaluation of VSR and its scheduling

For specified SR and CR, there might be a number of possible scheduling schemas, each of which has a different performance. In this section, we use three indicators to evaluate the performance of *VSR* and its scheduling.

4.1 Requirement satisfaction degree (RSD)

The degree to which the requirements in *CR* are satisfied by the scheduling of *VSR* is denoted by requirement satisfaction degree RSD(*VSR*). This is an important metric used in traditional service selection and composition approaches.

We use the ratio between the number of requirements satisfied by the *VSR* scheduling and the total number of requirements in *CR* as our measure of RSD:

$$\mathrm{RSD}(VSR) = \frac{\left|\bigcup_{AS\in Scheduling} AS.cr\right|}{|CR|}$$

Thus, RSD takes a value in the interval [0,1].

Note there is a pre-condition that, if $\{sr_1, \ldots, sr_k\}$ are allocated to satisfy the requirement *cr*, then all resources in $\{sr_1, \ldots, sr_k\}$ could satisfy *cr*'s QoS and price expectations. This is ensured during the resource selection and scheduling algorithms in the next section but is not considered in the measurement of RSD.

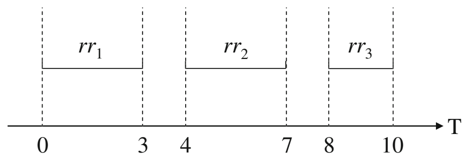

Fig. 4 An example of TS-type scheduling

4.2 Service utilization rate (SUR)

SUR(*VSR*) measures the average utilization rate of the resources contained in *VSR*.

We first give a definition of SUR according to sharing type. For a given $sr_j \in SR$, we let W $= \{AS|AS \in Scheduling \wedge AS.sr = sr_j\}$ as all the scheduling of the resource sr_j.

If $sr_j.ShareType =$ TS, the utilization rate is measured by the ratio between the length of the period occupied purely by the scheduling and the total duration that sr_j is occupied:

$$\text{SUR}\left(sr_j\right) = \frac{\sum_{AS_i \in W} AS_i.D}{\max\{AS_i.ST + AS_i.D\} - \min\{AS_i.ST\}}.$$

For example, in Fig. 4, there are three requirements allocated to one TS-type resource. The total period is 10 hours and the period occupied by the scheduling is $(3-0)+(7-4)+(10-8) = 8$, and during the period [3,4] and [7,8], the resource is free. So SUR $= 8/10 = 0.8$.

If $sr_j.ShareType =$ SS, the utilization rate is measured by the ratio between the total space occupied by the scheduling and the total available space:

$$\text{SUR}\left(sr_j\right) = \frac{\sum_{AS_i \in W} AS_i.Q^S}{|W|}$$

If $sr_j.ShareType =$ ES, there is no any sharing or waste, so SUR$(sr_j) = 1$.

If $sr_j.ShareType =$ CS, the measurement is a little more complex because at different time points, there might be different numbers of requirements concurrently using sr_j. Let $TM = \bigcup_{AS_i \in W}\{AS_i.ST, AS_i.ST + AS_i.D\}$ be the set of all start and end time points in the scheduling of sr_j. Suppose that there are a total of p elements $\{t_1, t_2, \ldots, t_p\}$ in *TM* and that they are sorted in ascending order. Then, SUR is measured for different time periods and synthesized together as a proportional sum weighted by the length of each time period:

$$\text{SUR}\left(sr_j\right) = \sum_{i=1}^{p-1}\left(\frac{|\{AS\,|AS \in W \wedge AS.\,ST \le t_i \wedge (AS.ST + AS.D) \ge t_{i+1}\}|}{sr_j.MC} \times \frac{t_{i+1} - t_i}{\max TM - \min TM}\right).$$

Taking Fig. 5 as an example, suppose that s_j is assigned to five requirements and that there are eight time points in T.

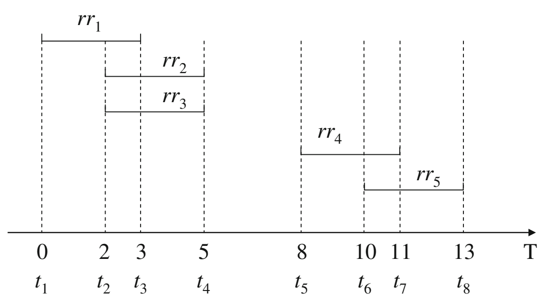

Fig. 5 An example of CS-type scheduling

Suppose the *maximal concurrency number (MC)* of s_j is 3, so SUR $\left(sr_j\right) = \frac{1}{3} \times \frac{2}{13} + \frac{3}{3} \times \frac{1}{13} + \frac{2}{3} \times \frac{2}{13} + \frac{0}{3} \times \frac{3}{13} + \frac{1}{3} \times \frac{2}{13} + \frac{2}{3} \times \frac{1}{13} + \frac{1}{3} \times \frac{2}{13} \approx 0.385$. The first $\frac{1}{3}$ means that during $t_1$and t_2, there is only one requirement (rr_1) occupying the resource, so the resource utilization is 1/3. The second $\frac{2}{13}$ means that the ratio of the period $[t_1, t_2]$ in the whole service duration is (2-0)/(13-0). The meanings of other fractions are similar.

Combining the SUR of all the resources, SUR $(VSR) = \frac{\sum_{sr_j \in SR} \text{SUR}(sr_j)}{|SR|}$ is the average of all the resources' SUR and lies in the interval [0,1].

4.3 Service sharing cost (SSC)

As has already been discussed, the resource utilization rate can be improved by sharing of service resources. However, this also increases the cost of coordination between multiple requirements that use the same resource. During resource selection and scheduling, it is necessary to consider the sharing cost as an optimization objective, which we denote by service sharing cost SSC(*VSR*).

Different sharing types lead to different definitions of SSC. For a given $sr_j \in VSR.SR$, we let $W = \{AS|AS \in Scheduling \wedge AS.sr = sr_j\}$.

- If $sr_j.ShareType =$ ES, there is no sharing, so SSC$(sr_j) = 0$.
- If $sr_j.ShareType =$ CS, although multiple requirements might share sr_j at some time, they are concurrently satisfied by sr_j and there are no any coordination between these requirements, so SSC$(sr_j) = 0$.
- If $sr_j.ShareType =$ TS, sr_j will be switched between multiple requirements over the time period, which may take extra time (for example, a vehicle moving from one customer's location to another's), so the more requirements that sr_j is allocated to, the greater the potential cost incurred. For simplicity, we use the number of requirements that sr_j is allocated to as our measure of SSC, that is, SSC$(sr_j) = |W|$.

- If $sr_j.ShareType = $ SS, the situation is the same as for TS, and so SSC(sr_j) = $|W|$.

The total SSC is the calculated as $\mathrm{SSC}(CR, VSR) = \frac{\sum_{sr_j \in SR} SSC(CR, sr_j)}{|SR|}$ being an integer.

5 A sharing-based service selection and scheduling approach

5.1 Optimization objectives, variables and process

Given a set of requirements $CR = \{cr_1, \ldots, cr_n\}$ and a set of available resources $SR = \{sr_1, \ldots, sr_m\}$, we select several resources from *SR* and allocate them to requirements or parts of requirements in *CR*, thereby constructing a schema $VSR = (CR, SR, Scheduling)$.

As mentioned above, there are three optimization objectives for such sharing-based service selection and scheduling:

Max RSD(CR, VSR),

Max SUR(CR, VSR), and

Min SSC(CR, VSR).

This is a typical multi-objective optimization problem [27].

The solution to the problem is quite complex and involves the allocation of resources to requirements specified according to both time and quantity. To simplify the complexity of solving the problem, we use the combination of the following four decision variables to represent a feasible solution:

1. $x_i = 0/1\ (i = 1, \ldots, n)$ indicates whether or not the ith requirement is satisfied;
2. $y_{ij} = 0/1\ (i = 1, \ldots, n,\ j = 1, \ldots, m)$ indicates whether the ith requirement is satisfied by the jth resource; for each i, there may be multiple j with y_{ij}=1, i.e., multiple resources could be allocated to one requirement;
3. $p_{ij} = [ST_{ij}, D_{ij}]$ gives the start time and duration that the jth resource is used to satisfy the ith requirement. It is a representation of resource allocation over time;
4. $z_{ij} = 1$ or a decimal fraction indicates the proportion of the jth resource that is allocated to the ith requirement. It is a representation of resource allocation over quantity.

Different resource sharing types use different combinations of these decision variables to represent a feasible schedule and to evaluate the three optimization objectives, as shown in Table 2.

For example, for ES-type resources, the scheduling is not related to time and quantity, so the decision variables are only $\{x_i\}, \{y_{ij}\}$; for SS-type resources, the scheduling is solely related to quantity but not related to time, so the decision variables are $\{x_i\}, \{y_{ij}\}, \{z_{ij}\}$. The right part of Table 2 summarizes the variables for calculating the three optimization objectives (RSD, SSC, and SUR) aiming at different sharing types, and based on the discussions in Sect. 4, readers can easily get such results.

Table 2 Relationships between decision variables and optimization objectives

Sharing type	Variables for representing a schedule	Variables for calculating the optimization objectives		
		RSD	SSC	SUR
ES	$\{x_i\}, \{y_{ij}\}$	$\{x_i\}$	0	1
CS	$\{x_i\}, \{y_{ij}\}, \{p_{ij}\}$	$\{x_i\}$	0	$\{y_{ij}\}, \{p_{ij}\}$
SS	$\{x_i\}, \{y_{ij}\}, \{z_{ij}\}$	$\{x_i\}$	$\{y_{ij}\}$	$\{z_{ij}\}$
TS	$\{x_i\}, \{y_{ij}\}, \{p_{ij}\}$	$\{x_i\}$	$\{y_{ij}\}$	$\{p_{ij}\}$

A basic constraint on scheduling is that, when there are multiple resource requirements (*rr*) in a *cr*, either all of these should be satisfied in a feasible VSR or else none of should be satisfied. That is, a *cr* must be considered in its entirety.

It is clear that the search space for this problem is quite big and that it is an NP problem. We use a genetic algorithm (GA) to search for the optimal solution, and to simplify the operations in the GA algorithm we adopt a three-level nested algorithm to optimize the four decision variables. Specifically, the first level deals with optimizing $\{x_i\}$, the second level deals with optimizing $\{y_{ij}\}$ and the third level deals with optimizing $\{z_{ij}\}$ or $\{p_{ij}\}$. The algorithm is briefly described in Fig. 6.

Firstly, the 1st-level algorithm deals with $\{x_i\}$, i.e., to randomly select a set of requirements that will be satisfied, calculate the RSD according to $\{x_i\}$ (from Table 2 we see that RSD is only related to $\{x_i\}$), then enter the 2nd-level's optimization. Based on the results returned by the 2nd-level algorithm, it calculates RSD*SUR/SSC, checks whether the exit condition is satisfied. If it is not satisfied, it enters the next loop and looks for better result and finally returns the optimized scheduling result. The process of the 2nd-level algorithm is quite similar as the 1st-level one, except that it deals with $\{y_{ij}\}$(to select a set of resources for each requirement that should be satisfied) and calculates SSC according to $\{y_{ij}\}$, then enters the 3rd-level's optimization. The 3rd-level algorithm deals with $\{z_{ij}\}$ or $\{p_{ij}\}$, i.e., to specify the time and space allocation between the selected resources (represented by $\{y_{ij}\}$) and the selected requirements (represented by $\{x_i\}$), calculates and optimizes SUR, then returns the optimized scheduling having maximal SUR back to the 2nd-level algorithm, which further returns the optimized scheduling have maximal SUR/SSC to the 1st-level algorithm. Finally, the scheduling with maximal (RSD*SUR/SSC) is found. The arrows in Fig. 6 show the execution steps between the three levels of algorithms.

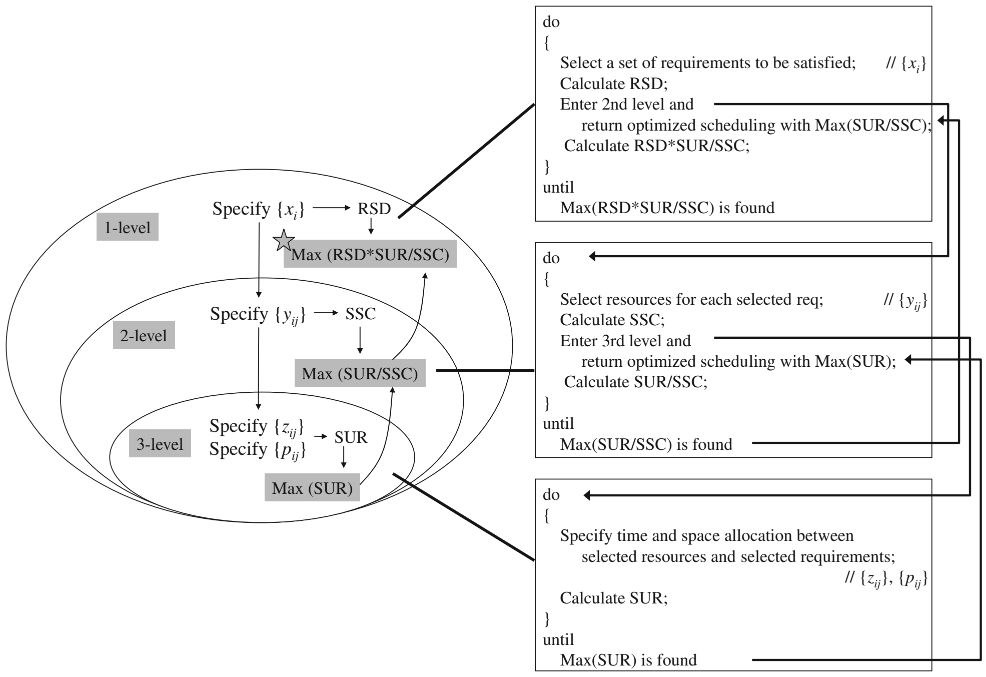
1-level
Specify {x_i} → RSD
Max (RSD*SUR/SSC)
2-level
Specify {y_ij} → SSC
Max (SUR/SSC)
3-level
Specify {z_ij}
Specify {p_ij} → SUR
Max (SUR)
do { Select a set of requirements to be satisfied; // {x_i} Calculate RSD; Enter 2nd level and return optimized scheduling with Max(SUR/SSC); Calculate RSD*SUR/SSC; } until Max(RSD*SUR/SSC) is found
do { Select resources for each selected req; // {y_ij} Calculate SSC; Enter 3rd level and return optimized scheduling with Max(SUR); Calculate SUR/SSC; } until Max(SUR/SSC) is found
do { Specify time and space allocation between selected resources and selected requirements; // {z_ij}, {p_ij} Calculate SUR; } until Max(SUR) is found

Fig. 6 Basic process of the three-level nested algorithm

5.2 Global Genetic Algorithm (GGA)

We first give the first level of the algorithm, called the Global Genetic Algorithm (GGA).

```
GGA (Global Genetic Algorithm)
Input: CR, SR
Output: $VSR = \langle CR, SR, Scheduling\rangle$
Step 1: Specify the population size NP, the maximum number of generations NG, crossover probability $p_c$ and mutation probability $p_m$, and set g=1 to be the number of the population.
Step 2: Randomly generate an initial population $P_g(X)$ in which each chromosome is represented by $X = [x_1, x_2, \ldots, x_n], x_i = 0/1, n = |CR|$.
Step 3: For each X ∈ $P_g(X)$, repeat Steps 4 ∼ 6.
{
    Step 4: $\overline{CR} = \{cr_i | x_i = 1\}$, $\overline{RR} = \cup_{cr_i \in \overline{CR}} cr_i.RR$.
    Step 5: Group the elements in $\overline{RR}$ according to the required resource type (that is, $rr.T$), with $\overline{RR_k}$ representing the set of resource requirements of type $T_k$(k=1, 2, …, p). Also, group the resources in SR according to their type, with the resources in $\overline{SR_k}$ being the candidates for satisfying the requirements in $\overline{RR_k}$.
    Step 6: For each $\overline{RR_k}$ and the corresponding $\overline{SR_k}$:
    {
      Switch(The sharing type of the required resources in $\overline{RR_k}$)
          Case "ES": invoke algorithm ESA($\overline{RR_k}$, $\overline{SR_k}$);
          Case "CS": invoke algorithm CSA($\overline{RR_k}$, $\overline{SR_k}$);
          Case "SS": invoke algorithm SSA($\overline{RR_k}$, $\overline{SR_k}$);
          Case "TS": invoke algorithm TSA($\overline{RR_k}$, $\overline{SR_k}$);
          Each algorithm above returns a set of scheduling schemas, denoted by Scheduling($\overline{RR_k}$, $\overline{SR_k}$). If there exists at least one k that makes Scheduling($\overline{RR_k}$, $\overline{SR_k}$) = Ø, then the chromosome X chosen in Step 3 is invalid because there is no possible feasible scheduling that could satisfy X. In this case, set X's fitness to be $F(X) = 0$, return Step 3 and choose another chromosome.
    }
    If Scheduling ($\overline{RR_k}$, $\overline{SR_k}$) ≠ Ø for all k, then X's final scheduling schema is $\bigcup_{k=1}^{p} Scheduling\left(\overline{RR_k}, \overline{SR_k}\right)$.
    For $\bigcup_{k=1}^{p} Scheduling\left(\overline{RR_k}, \overline{SR_k}\right)$, calculate the fitness of X to be $F(X) = \frac{RSD \times SUR}{SSC}$.
}
Step 7: Find the optimal chromosome X* that has the maximal fitness over the current and all previous generations.
Step 8: If $g + 1 > NG$, go to Step 10, otherwise go to Step 9.
Step 9: $g = g+1$; perform the selection, crossover, and mutation to generate the next population $P_{k+1}$(X), then go to Step 3.
Step 10: Output X* and its corresponding scheduling schema.
```

In GGA, a chromosome X represents a specific scheduling schema indicating what requirements are to be satisfied, but only if the second-level algorithm (Step 6) returns a valid scheduling, is X valid. RSD is calculated from X itself, while SUR and SSC are calculated from the valid scheduling for X. X's fitness is then calculated by $\frac{RSD \times SUR}{SSC}$.

In Step 9, we use the roulette wheel selection scheme to determine which chromosomes should be inherited by the next generation, the one-cut-point strategy for crossover and a simple mutation strategy for mutation.

5.3 Exclusive-Sharing-Oriented Algorithm (ESA)

```
ESA (Exclusive-Sharing-Oriented Algorithm)
Input: RR, SR
Output: Scheduling(RR, SR)
Step 1: For each $rr_i \in RR$
{
    Invoke algorithm RSA($rr_i$, $SR$) to select a candidate service resource set $\overline{SR_i}$ in which each sr satisfies the QoS and price constraints in $rr_i$.
    If $\overline{SR_i} = \emptyset$, then there are no available resources that satisfy $rr_i$. Set Scheduling $(RR, SR) = \emptyset$ and exit this algorithm.
}
Step 2: There might exist $i, j$ that make $\overline{SR_i} \cap \overline{SR_j} \neq \emptyset$, i.e., some resources could satisfy both $rr_i$ and $rr_j$. Because resources in SR are all exclusive sharing (i.e., each $rr_i$ must be allocated one resource to), $|\bigcup_{rr_i \in RR} \overline{SR_i}| < |RR|$ indicates that there are not enough candidate resources to satisfy all the requirements. Set Scheduling $(RR, SR) = \emptyset$ and exit. Otherwise go to Step 3.
Step 3: For exclusive-sharing resources, SUR and SSC are not related to Scheduling(RR, SR) (from Table 2, we see that SUR=1 and SSC=0 always), so the problem is transformed into: for each $rr_i \in RR$, select one $sr_j$ from $\overline{SR_i}$ that gives the most cost-effective solution.
The optimization objective is

$$\max \sum_{i=1}^{n} \sum_{j=1}^{m} \left( y_{ij} \times \frac{sr_j.QoS}{sr_j.P} \right)$$

such that:
• $y_{ij} = 0/1$ indicates whether or not the $j$th resource ($sr_j$) is allocated to satisfying the $i$th requirement ($rr_i$);
• if $y_{ij} = 1$, then $sr_j \in \overline{SR_i}$: the selected resource must satisfy the QoS and price constraints for the requirement;
• for all $i$, $\sum_{j=1}^{m} y_{ij} = 1$: one and only one resource is allocated to a requirement;
• for all $j$, $\sum_{i=1}^{n} y_{ij} \leq 1$: a resource is allocated to at most one requirement.
The 0–1 integer linear programming approach is adopted to solve this problem and obtain Scheduling(RR, SR). We do not give details of this sub-algorithm here.
Note that it is not necessary to enter the 3rd-level algorithm to schedule the time and allocate the quantity because the scheduling of exclusive-sharing resources is not related to either of them.
```

5.4 Concurrent-Sharing-Oriented Algorithm (CSA)

```
CSA (Concurrent-Sharing-Oriented Algorithm)
Input: RR, SR
Output: Scheduling(RR, SR)
Step 1: For each $rr_i \in RR$
{
    Invoke the algorithm RSA($rr_i$, $SR$) to select a candidate service resource set $\overline{SR_i}$ in which each sr satisfies the QoS and price constraints for $rr_i$.

    If $\overline{SR_i} = \emptyset$, then there are no feasible resources that satisfy $rr_i$, so set Scheduling $(RR, SR) = \emptyset$ and exit this algorithm.
}
Step 2: Specify NP, NG, $p_c$, and $p_m$ for the second-level GA.
Step 3: Let g=1 be the number of current generation. Randomly generate an initial population $P_g(Y)$ in which a chromosome is represented by Y=$\begin{bmatrix} y_{11} & \cdots & y_{1m} \\ y_{21} & & y_{2m} \\ \vdots & & \vdots \\ y_{n1} & \cdots & y_{nm} \end{bmatrix}$ ($n = |CR|, m = |SR|$), where $y_{ij}$=0/1 indicates whether or not $rr_i$ is satisfied by $sr_j$ in the corresponding schedule.
A chromosome is valid only if the following constraints are satisfied:

• if $y_{ij}$=1, then $sr_j \in \overline{SR_i}$: the selected resource must satisfy the QoS and price constraints in the requirement;
• for arbitrary $i$, $\sum_{j=1}^{m} y_{ij} = 1$: one and only one resource is allocated to a requirement.
Step 4: For each Y $\in P_g(Y)$
{
  Step 5: For each resource $sr_j \in SR$
  {
    Step 6: For each $rr_i$ that satisfies $y_{ij}$=1
    {
        Calculate $K = |\{AS|AS \in$ Scheduling$(RR, SR) \wedge$ AS.$sr = sr_j \wedge$ [AS.ST, AS.ST + AS.D] $\cap$ [$rr_i.LST, rr_i.$ LST + $rr_i$.D] $\neq \emptyset\}|$ be the total number of requirements that $sr_j$ has been allocated to in the expected period of $rr_i$.
        If $K < sr_j$.MC (indicating that $sr_j$ has not yet reached its maximal concurrency number), then construct a schedule $AS_i = [rr_i, sr_j, 1, rr_i.LST, rr_i.D]$ and add it to Scheduling(RR, SR).
        Else if $K \geq sr_j$.MC, $rr_i$ cannot be satisfied by $sr_j$ and hence Y is invalid. Exit from Step 5 and 6, go to Step 4 and choose another chromosome.
    }
  }

  Step 7: According to Scheduling(RR, SR), calculate the fitness of each chromosome Y by $F(Y)$=SUR.
}
Step 8: Find the optimal chromosome Y* that has the maximum fitness over the current and all previous generations.
Step 9: If $g+ > NG$, go to Step 11, otherwise go to Step 10.
Step 10: $g = g + 1$; perform the selection, crossover, and mutation to generate the next population $P_{g+1}$(Y), then go to Step 4.
Step 11: Output Y* and the corresponding scheduling schema.
```

Step 1 uses RSA to compress the search space for each requirement *rr*. Steps 2 to 11 are the second-level optimization process. Step 3 constructs the population, where each chromosome being a matrix represents a resource-requirement allocation schema (a many-to-many relationship). For each allocation schema, Steps 5 and 6 allocate each requirement to a selected resource one at a time, ensuring that the total number of requirements allocated to one resource does not exceed the resource's maximal concurrency number Steps 7 and 8 look for the optimal allocation solution by calculating and comparing the fitness of each chromosome.

In Step 10, the roulette wheel selection scheme is used to determine which chromosomes should be inherited by the next generation. The one-cut-point strategy is used for crossover, that is, each chromosome is partitioned into two parts (k and $m - k$), and these parts are re-matched to form two new chromosomes. This is shown in Fig. 7.

For the mutation, we use a swap mutation strategy. That is, we randomly select two genes (y_{ij} and y_{ik}) from the same row and swap the values of y_{ij} and y_{ik}, with the constraint that sr_j and sr_k should both belong to $\overline{SR_i}$.

Note that the crossover operation may produce invalid chromosomes (that is, $\exists i$, $\sum_{j=1}^{m} y_{ij} = 0$ or $\sum_{j=1}^{m} y_{ij} > 1$), and these need to be repaired. The repairing process is:

- if $\sum_{j=1}^{m} y_{ij} = 0$, randomly select one sr_j from $\overline{SR_i}$ and set y_{ij}=1;
- if $\sum_{j=1}^{m} y_{ij} > 1$, randomly select one sr from $\{sr_j | y_{ij} = 1\}$ and set y_{ij}=0 for each $sr_j \in \{sr_j | y_{ij} = 1 \wedge sr_j \neq sr\}$.

5.5 Space-Sharing-Oriented Algorithm (SSA)

```
SSA (Space-Sharing-Oriented Algorithm)
Input: RR, SR
Output: Scheduling(RR, SR)
Step 1: For each $rr_i \in RR$
{
    Invoke the algorithm RSA($rr_i$, $SR$) to select a candidate service
    resource set $\overline{SR_i}$ in which each sr satisfies the QoS and price
    constraints in $rr_i$.
    If $\overline{SR_i} = \emptyset$, then there are no feasible resources that satisfy $rr_i$.
    Set Scheduling $(RR, SR) = \emptyset$ and exit this algorithm.
}
Step 2: For each $\overline{SR_i}$, $|\overline{SR_i}| < \lceil rr_i.Q^R \rceil$ indicates the quantity of
required resources in $rr_i$ that cannot be satisfied by $\overline{SR_i}$. Set Scheduling$(RR, SR) = \emptyset$ and exit. Otherwise go to Step 3.
Step 3: Sort the elements in $RR = \{rr_i\}$ in ascending order according
to $\frac{rr_i.QoS}{rr_i.Price}$. This is to ensure that the requirements that could be most
easily satisfied are dealt with earlier in the following steps.
Step 4: For each $rr_i$
{
    Sort the elements in $\overline{SR_i}$ in ascending order according to $\frac{sr_j.QoS}{sr_j.Price}$.

    If $|\overline{SR_i}| < \lfloor rr_i.Q^R \rfloor$, then let Scheduling $(RR, SR) = \emptyset$ and
    exit this algorithm.
    Select the first $\lfloor rr_i.Q^R \rfloor$ resources (denoted by $SR_i^{(IP)}$) from
    $\overline{SR_i}$ and allocate them to $rr_i$. For each selected resource, construct the corresponding scheduling and add it to Scheduling(RR,
    SR).
    For each $rr_k \in RR$ with $k \neq i$, update $\overline{SR_k}$ by $\overline{SR_k} =
    \overline{SR_k} \backslash SR_i^{(IP)}$.
}
Step 5: Denote $RR^{(DP)} = \{rr_i | rr_i \in RR \wedge (rr_i.Q^R - \lfloor rr_i.Q^R \rfloor) > 0\}$.
For each $rr_i \in RR^{(DP)}$, if $\overline{SR_i} = \emptyset$ then set Scheduling $(RR, SR) =
\emptyset$ and exit this algorithm.
Step 6: Specify NP, NG, $p_c$ and $p_m$ for the second-level GA.
Let g=1 be the number of the current generation. Randomly generate
an initial population $P_g(Y)$ in which each chromosome is represented
by $Y = \begin{bmatrix} y_{11} & \cdots & y_{1m} \\ y_{21} & & y_{2m} \\ \vdots & & \vdots \\ y_{n1} & \cdots & y_{nm} \end{bmatrix}$. $y_{ij}$=0/1 indicates whether or not $rr_i$ is satis-
fied by $sr_j$. $n = |RR^{(DP)}|$, $m = \left|\bigcup_{rr_i \in RR^{(DP)}} \overline{SR_i}\right|$, so that the length
of a chromosome is $n \times m$.
    A chromosome is valid only if the following constraints are
    satisfied:

    • for each $i$, $\sum_{j=1}^{m} y_{ij} \geq 1$ and $\left(rr_i.Q^R - \lfloor rr_i.Q^R \rfloor\right) \geq
    \sum_{y_{ij}=1} sr_j.MSU$: there is at least one resource allocated to each
    requirement, and if there are multiple resources allocated to one
    requirement, each of them should have the chance of contributing
    at least its MSU (minimal sharing unit)'s space to satisfying the
    requirement;
    • if $y_{ij}$=1, then $sr_j \in \overline{SR_i}$: the selected resource must satisfy
    the QoS and price constraints for the requirement;
    • for each $j$, if $\sum_{i=1}^{n} y_{ij} \geq 1$, then $\sum_{i=1}^{n} y_{ij} \leq \frac{1}{sr_j.MSU}$: the
    number of requirements satisfied by each resource should not
    exceed the maximal number $\frac{1}{sr_j.MSU}$.
Step 7: For each $Y \in P_g(Y)$
{
    Under the resource allocation scheme defined by Y, the
    problem of searching for the optimal quantity allocation is
    described by: for each $rr_i \in RR^{(DP)}$, select $sr_j$ (with quan-
    tity $z_{ij}$) from $\overline{SR_i}$ that gives the most cost-effective final solution.

    The optimization objective is

        $$\max \sum_{i=1}^{n} \sum_{j=1}^{m} \left( z_{ij} \times \frac{sr_j.QoS}{sr_j.P} \right),$$

    such that:

        • $0 \leq z_{ij} \leq 1$ indicates the quantity of the $j$th resource ($sr_j$)
        that is allocated to satisfying the $i$th requirement ($rr_i$);
        • if $y_{ij}$=1, then $z_{ij} = L \times sr_j.MSU$ ($L \geq 1$ is an integer):
        the quantity of a resource allocated to each requirement
        should be multiple of its minimal sharing unit;
        • for each $j$, $\sum_{y_{ij}=1} z_{ij} \leq 1$: the total allocation of each
        resource should not exceed 1;
        • for each $i$, $\sum_{y_{ij}=1} z_{ij} = r_{ij}.Q^R$: the total quantity of
        resources to which a requirement is allocated should be
        equal to the requested quantity.

    A linear programming approach is adopted for solving this prob-
    lem to obtain Scheduling(RR, SR). If Scheduling $(RR, SR) = \emptyset$,
    then Y is invalid, go back to Step 7 and choose another chromo-
    some. Otherwise, calculate the fitness of $\underline{Y}$ by $F(Y)$=SUR/SSC.
}
Step 8: For the current optimal solution, choose Y* with the maximum
fitness and whose scheduling schema is not empty from the current and
all previous generations.
Step 9: If $g + 1 > NG$, go to Step 11, otherwise go to Step 10.
Step 10: $g = g$+1; perform the selection, crossover and mutation to
generate the next population $P_{g+1}$(Y), then go to Step 7.
Step 11: Output Y* and the corresponding Scheduling(RR, SR).
```

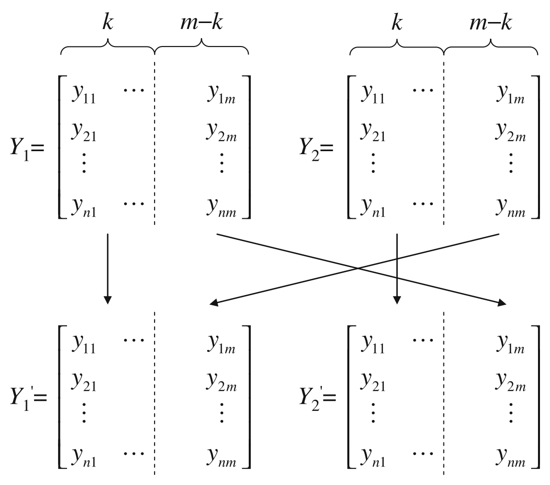

Fig. 7 One-cut-point crossover strategy

Step 1 of SSA is the same as CSA, designed to reduce the search space. Steps 2 to 4 deal with the integer part of the quantity of requested resources for each requirement using a greedy policy. Steps 5 to 11 are the second-level optimization process for allocating resources to the decimal part of each requirement. Step 5 checks whether the quantity of the remaining candidate resources is sufficient for the requirements. Step 6 constructs the population, where each chromosome represents a resource-requirement allocation schema. For the allocation schema defined by each chromosome, Step 7 is the third-level optimization for obtaining the concrete allocation quantity of each resource.

In Step 7, the fitness of Y is calculated as $F(Y)$=SUR/SSC. It is more complex in calculating SUR and SCC of space-sharing resources because the integer and decimal part of each requirement should be jointly considered. Specifically, SSC=$\frac{\sum_{j=1}^{m}\sum_{i=1}^{n} y_{ij}}{|\{j|\exists i, y_{ij}=1\}|+\sum_{i=1}^{n}\lfloor rr_i.Q^R\rfloor}$ where $\sum_{i=1}^{n} y_{ij}$ in the numerator calculates the SSC of resource sr_j and $\sum_{j=1}^{m}\sum_{i=1}^{n} y_{ij}$ is the sum of SSC for all the resources, and $\left|\left\{j\,|\exists i,\, y_{ij}=1\right\}\right|$ in the denominator calculates the total number of resources in the scheduling of the decimal part of requirements and $\sum_{i=1}^{n}\lfloor rr_i.Q^R\rfloor$ is the total number of resources in the scheduling of the integer part of requirements. SUR=$\frac{\sum_{i=1}^{n} rr_i.Q^R}{|\{j|\exists i, y_{ij}=1\}|+\sum_{i=1}^{n}\lfloor rr_i.Q^R\rfloor}$, where the numerator is the sum of the resource quantity requested in each requirement (also the sum of SUR for all allocated resources in the scheduling represented by Y) and the denominator is the total available space (calculated by the total number of allocated resources in the scheduling represented by Y).

5.6 Time-Sharing-Oriented Algorithm (TSA)

TSA is quite similar to CSA, i.e., both are to allocate one resource to multiple requirements along the time. Actually, a time-sharing service resource may be regarded as a concurrent sharing one whose maximal concurrency number (MC) is 1. Besides, the optimization objective will be SUR/SCC instead of pure SUR.

Therefore, we do not give the full TSA here but to make a minor revision on CSA. The modifications are emphasized by the bold font below.

Line 2 of Step 6: If ***K*= *0*** (**indicating that *sr_j* has not yet been occupied during the expected period of *rr_i***), then construct the scheduling $AS_i = [rr_i,\, sr_j,\, 1,\, rr_i.LST,\, rr_i.D]$ and add it to Scheduling(*RR, SR*).
Step 7: According to Scheduling(*RR, SR*), calculate the fitness of each chromosome Y as $\boldsymbol{F(Y)} = \mathbf{SUR/SSC}$;

5.7 Resource Selection Algorithm (RSA)

In the four algorithms detailed above is the common sub-algorithm RSA(rr, SR). This algorithm checks each resource in *SR* whether it could satisfy *rr*'s QoS and price expectations, then returns all the qualified resources.

```
RSA(Resource Selection Algorithm)
Input: $rr, SR$
Output: $\overline{SR}$
Let $\overline{SR} = \emptyset$;
For each $sr \in SR$
{
    If ($sr$.P >= $rr$.LP and $sr$.P <= $rr$.HP and $sr$.QoS >= $rr$.LQ
    and $sr$.QoS <= $rr$.HQ)
    Then $\overline{SR} = \overline{SR} \cup \{sr\}$;
}
```

This algorithm is not the focus of this paper, so it keeps simple and the resource selection criteria keep quite strict, i.e., only if the actual QoS and price of a resource fall in the expectation scope of the customer requirement, could it be selected as a candidate.

6 Experiments and comparisons

6.1 Experiment settings

In this section, we use several experiments to compare our approach with the traditional non-sharing service selection approach. A real-world ocean transportation service [4] is

Table 3 Typical resource types in OTS

Resource type	Mode of existence	Services	Sharing type
Ship cabins	Physical entities	Container loading	ES
Vehicles	Physical entities	Land transportation	TS
Containers	Physical entities	Goods loading	SS
Customs declaration service	Web services	Customs declaration	CS

used as the experiment background in which a service broker obtains service resources from multiple service providers (shipping companies, container agencies, truck companies and dock fields) and provides an integrated service to the consigners. The resources employed in this service are listed in Table 3.

Ship cabins are the space for storing containers in the ship during transportation. If a cabin is occupied by a container from a specific consigner, then it cannot be allocated to other consigners for the duration of transportation and this is thus a case of exclusive sharing. Vehicles are used for transporting containers from warehouses to the dock and can be time-share allocated to multiple customers. Containers are of course space-sharing resources. Customs declaration services are web services for receiving the customs declaration applications and sending inspection results to the applicants and thus are concurrent-shared with a maximal concurrency number constraint.

Firstly, we briefly introduce the non-sharing service selection approach. As discussed in Sect. 1 and shown in Fig. 1, the non-sharing approach deals with requirements one by one without considering any sharing between multiple requirements. No matter what sharing type a service resource has, it is always scheduled by an exclusive-usage way. For each resource requirement rr_i, the algorithm RSA(rr_i, SR) is simply invoked to select a candidate service resource set $\overline{SR_i}$ ($|\overline{SR_i}| = \lceil rr_i.Q^R \rceil$) in which each sr satisfies the QoS and price constraints in rr_i. If no enough $\overline{SR_i}$ could be found, the customer requirement cr that rr_i belongs to cannot be satisfied. After all the customer requirements have been repeatedly dealt with and the scheduling result has been obtained, the method in Sect. 4 is applied to evaluate requirement satisfaction degree (RSD), service sharing cost (SSC), and service utilization rate (SUR) of the scheduling.

We conducted three experiments for comparison. An outline of the experiments is shown in Table 4, including the experiment settings, purpose, data preparation, comparison plan, and the metrics used for analysis and comparison.

6.2 Experiment 1

In the first experiment, 40 instances of the above-mentioned four types of resources were randomly generated. The resource price is in the range between 20 and 30, and the QoS value is in the range between 0 and 1. For customs declaration services, their maximal concurrency number (MC) is between 5 and 10; for containers, their minimal sharing unit (MSU) is 0.1; and for vehicles, their maximal available period is between 8 am and 8 pm every day.

We further generated different numbers of customer requirements ($n = 10, 20, \ldots, 80$). Their QoS value is in the range between 0.3 and 0.9, and their price expectation is in the range between 21 and 29; therefore, some of them cannot be satisfied by the available resources. For the resource requirements requesting containers, the requested quantity is in the range of 0.2–4 and is a multiple of 0.2. For those requesting vehicles and customs declaration services, the expected latest start time (LST) is in the range between 8 am and 17 pm and the duration might be 1, 2, or 3 h. Each customer requirement contains four resource requirements which request the four types of resources in Table 3, respectively.

Note that the pre-defined ranges of related parameters attached to both service resources and customer requirements are approximately extracted from the real-world ocean transportation services, but the specific values of each concrete resource and requirement are randomly generated and distributed in the pre-defined ranges.

We use both GGA and the traditional service selection algorithm (denoted as TA) to deal with the resource selection and scheduling and determine requirement satisfaction degree (RSD), service sharing cost (SSC), and service utilization rate (SUR) and then use TP = RSD × SUD/SSC to indicate the global performance. The comparison results are shown in Fig. 8.

From these figures, we see that in GGA the service utilization rate (SUR) is continuously improved and reaches to a maximum value as |CR| increases. This is because resource sharing has been maximized and no more requirements can be satisfied by the sharing mechanism. Similarly, service sharing cost (SSC) also increases as |CR| increases until it becomes stable. Requirement satisfaction degree (RSD) decreases as |CR| increases. Initially, this decrease is slow because the available resources are relatively plentiful compared with the number of requirements, but once service utilization rate reaches its maximum, service sharing cost drops sharply. The global performance shows a similar trend.

Because the traditional approach does not allow for resource sharing, the service sharing cost is always 0 and service utilization rate is always low, leading also to low values for requirement satisfaction degree. The reason that service utilization rate is not 0 is that customs services can be shared in a concurrent manner.

Table 4 An outline of the experiments

Experiment no.	Experiment setting	Purpose	Data preparation	Comparison plan	Metrics
1	The quantity of available resources is fixed, and the number of customer requirements gradually increases	(1) To find the trends of RSD, SUR, SSD, and the global performance, with respect to the increase of the number of customer requirements;	(1) 40 instances of four types of resources in ocean transportation service are generated, and the values of their attributes are randomly distributed in the pre-defined range	There are totally eight times of experiments in which the number of customer requirements is 10, 20, …, 80, respectively	RSD SUR SSD TP = RSD × SUD/SSC
		(2) To compare the RSD, SUR, SSD and global performance between our approach and traditional non-sharing approach	(2) 10, 20, …, 80 customer requirements are separately generated, the values of whose attributes are randomly distributed in the pre-defined range	In each experiment, GGA and TA run once, separately	
				Then, the results of all the experiments are aggregated and the curves of RSD, SUR, SSD and TP with respect to the number of requirements are drawn	
2	The number of customer requirements is fixed, and the quantity of available resources gradually increases	(1) To find the trends of RSD, SUR, SSD, and the global performance, with respect to the increasement of the quantity of available resources;	(1) 20 customer requirements are generated;	There are totally eight times of experiments in which the quantity of available resources is 10, 20, …, 80, respectively	RSD SUR SSD TP = RSD × SUD/SSC
		(2) To compare the RSD, SUR, SSD, and global performance between our approach and traditional non-sharing approach	(1) 10, 20, …, 80 instances of resources are separately generated.	In each experiment, GGA and TA run once, separately	
				Then, the results of all the experiments are aggregated and the curves of RSD, SUR, SSD, and TP with respect to the quantity of resources are drawn	

Table 4 Continued

Experiment no.	Experiment setting	Purpose	Data preparation	Comparison plan	Metrics
3	The quantity of available resources and the number of requirements are both fixed, but the distribution of these requirements' arrival time changes	To find the potential impacts that the temporal distribution of requirements has on RSD, SUR, SSD and global performance	Two groups of requirements whose attribute settings are totally same. Requirements in the first group are uniformly distributed and the second is more concentrated	GGA runs twice for the two groups of requirements, respectively. Then the results of the two experiments are compared.	RSD SUR SSD TP = RSD × SUD/SSC

6.3 Experiment 2

In the second experiment, we generated 20 requirements for which the quantity requested and the QoS and price constraints were same as in the first experiment. We then generated different numbers of available resources ($m = 10, 20, \ldots, 80$) with the same QoS and price setup as in the first experiment. A comparison between GGA and TA is shown in Fig. 9.

From the figures we have the following observations. In GGA, the requirement satisfaction degree (RSD) increases as the number of available resources increases. Once the requirements that cannot be satisfied initially are satisfied by the increased number of resources, requirement satisfaction degree reaches a maximum. Service utilization rate (SUR) maintains a high level and is stable with slight fluctuations. Once requirement satisfaction degree reaches its maximum, service utilization rate remains stable because newly introduced resources no longer participate in satisfying the requirements. Service sharing cost (SSC) gradually decreases until it reaches a stable level. Overall, the global performance of GGA increases gradually until it reaches a maximum.

Although requirement satisfaction degree increases with the increase in resources in TA also, the rate of increase is significantly lower than in GGA because there is no sharing between requirements. In comparison with GGA, more resources are needed to satisfy the same number of requirements.

Another conclusion from the experiments is that when the available resources are sufficient relative to the number of requirements, requirement satisfaction degree remains high while service utilization rate and service sharing cost are both relatively low. On the contrary, when the available resources are insufficient, requirement satisfaction degree is low while both service utilization rate and service sharing cost are relatively high.

Note that in both the first and second experiments, the execution time for GGA is longer than for TA. The reason for this is that bundling multiple requirements and resource sharing increase the computation time. We do not show the figures here.

6.4 Experiment 3

In the third experiment, we consider the distribution of incoming requirements. We generate two groups of requirements with the same quantity and QoS and price constraints. As shown in Fig. 10 where the vertical axis is the number of arrival customer requirements at a specific time, the first group is uniformly distributed throughout the period between 8 am and 8 pm (i.e., every hour there are 3 arrival requirements), and the second one is more concentrated in the period

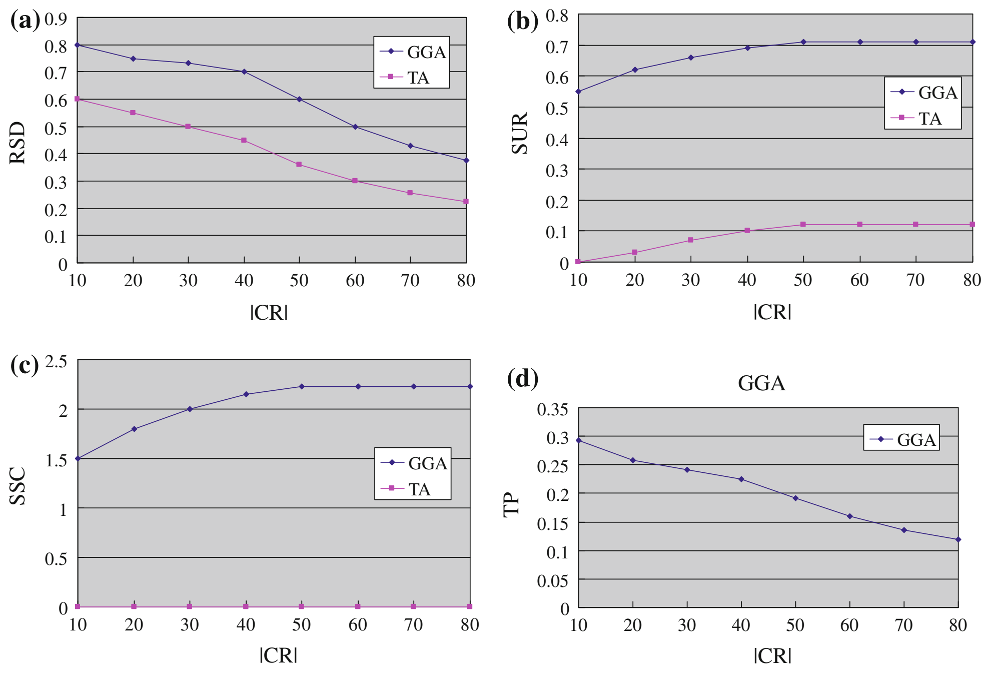

Fig. 8 Performance with respect to the number of requirements. **a** RSD w.r.t. |CR|, **b** SUR w.r.t. |CR|, **c** SSC w.r.t. |CR|, **d** Global performance w.r.t. |CR|

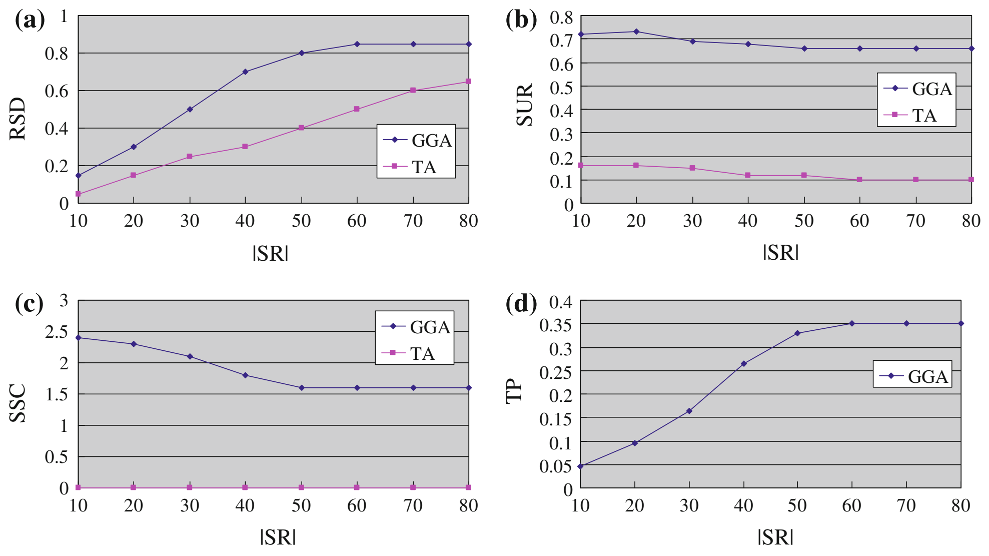

Fig. 9 Performance with respect to the number of available resources. **a** RSD w.r.t. |CR|, **b** SUR w.r.t. |CR|, **c** SSC w.r.t. |CR|, **d** Global performance w.r.t. |CR|

between 11 am and 2 pm (i.e., the number of arrival requirements are 7, 8, 8, and 7).

The results and comparison are shown in Fig. 11. From the figure, we see that, compared with temporally balanced requirement queue, the violent fluctuations of the requirement queue lead to significantly lower requirement satisfaction degree (RSD) and the global performance, but service utilization rate (SUR) and service sharing cost (SSC) remain about the same.

7 Conclusions and future work

In consideration of a typical scenario where large amounts of requirements are raised concurrently in an e-service environment, we suggest an approach to resource selection and scheduling based on requirement bundling and resource sharing. Resources are classified into four categories (exclusive-, time-, space-, and concurrent sharing) according to distinct features in the sharing patterns, and a nested three-level

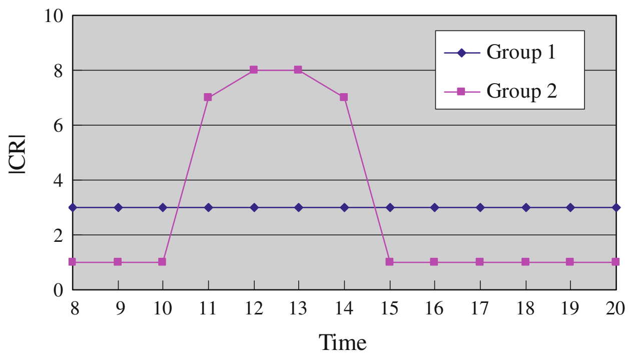

Fig. 10 Two groups of requirements with different temporal distributions

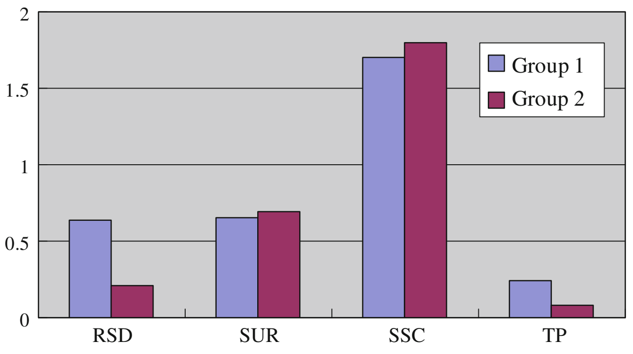

Fig. 11 Comparison between the performances for the two groups

algorithm is proposed to search for the optimal resource selection and scheduling solution for these multiple requirements. The effectiveness of this method is partially validated through a set of experiments.

Our approach will be of significance to typical modern service industries, such as production services, logistics services, travel services, and more specifically, the popular O2O(online-to-offline) e-business services where people reserve services from the website but access to the service in the real world. In these services, the resources are not only online softwarized ones but most of them are offline physical ones. Elaborate allocation and scheduling of these resources are quite required. Especially in the peak season, when many customer demands are raised at the same time, our approach ensures an improved tradeoff between requirement satisfaction degree (RSD), service utilization rate (SUR), and service sharing cost (SSC), thereby helping service providers make better decisions.

Applying our method to a specific service domain is not quite difficult. Most of the required data (including the basic information of resources and customer requirements) have already existed, and extra necessary data only include the sharing type of each type of resource and their corresponding minimal sharing unit, maximal available period, and maximal concurrency number.

Future research should consider the following points.

1. The three-level nested genetic algorithm adopted in our method is computationally expensive and might not obtain the global optimal solution because of the large search space and the random search policy. Improvements to this algorithm need to strike a balance between the time complexity and the optimality of the solution.
2. Resources that have features of several sharing patterns need a combined strategy for selection and scheduling.
3. The description models of both service requirement and service resource need further extension to be more in accord with real-life service scenarios, e.g., to refine the integrated QoS metrics into a set of fine-grained quality parameters, to define more complex but flexible expectations on QoS and price of service resources.

Acknowledgments This paper is a revised and extended version of the 2010 IEEE International Conference on e-Business Engineering (ICEBE 2010) paper "The Bundling of Multiple Requirements for Maximizing the Utilization of Service Resources". Research works in this paper are supported by the National Natural Science Foundation (NSF) of China (No. 60803091, 70971029, 61033005).

References

1. Gummerus J (2010) E-services as resources in customer value creation: a service logic approach. Manag Serv Qual 20(5): 425–439. doi:10.1108/09604521011073722
2. Skene J, Raimondi F, Emmerich W (2010) Service-level agreements for electronic services. IEEE Trans Softw Eng 36(2): 288–304. doi:10.1109/TSE.2009.55
3. Wang ZJ, Xu XF (2009) Bilateral resource integration service mode for value innovation. Comput Integr Manuf Syst 15(11):2216–2225
4. de Kinderen S, Gordijn J, Akkermans H (2006) Matching Complex Consumer Needs with e-Service Bundles. In: Proceedings of 19th BLED conference, June 2006, http://docs.e3value.com/bibtex/pdf/DeKinderenMatching2006.pdf
5. Wang ZJ, Xu XF, Chu DH, Ma C (2009) A case study on bi-lateral resource integration oriented marine logistics service system. In: Proceedings of 2009 IEEE international conference on service computing, Sept 2009, pp 458–465. doi:10.1109/SCC.2009.37
6. Gordijn J, de Kinderen S, Wieringa R (2008) Value-driven service matching. In: Proceedings of 16th IEEE international requirements engineering conference, Sept 2008, pp 67–70. doi:10.1109/RE.2008.10
7. Dustdar S, Schreiner W (2005) A survey on web services composition. Int J Web Grid Serv 1(1):1–30. doi:10.1504/IJWGS.2005.007545
8. Baida Z (2006) Software-aided service bundling: intelligent methods & tools for graphical service modeling. PhD Dissertation, SIKS Dissertation Series No. 2006-06, Vrije Universiteit, The Netherland
9. Wieder P, Seidel J, Wäldrich O, Ziegler W, Yahyapour R (2008) Using SLA for resource management and scheduling—a survey. In: Grid middleware and services. Springer, June 2008, pp 335–347. doi:10.1007/978-0-387-78446-5_22

10. Alrifai M, Risse T (2009) Combining global optimization with local selection for efficient QoS-aware service composition. In: Proceedings of 18th international conference on world wide web, April 2009, pp 881–890. doi:10.1145/1526709.1526828
11. Peer J (2005) Web service composition as AI planning—a survey. http://elektra.mcm.unisg.ch/pbwsc/docs/pfwsc.pdf
12. Cardellini V, Casalicchio E, Grassi V, Presti FL (2007) Flow-based service selection for web service composition supporting multiple QoS classes. In: Proceedings of international conference on web services, July 2007, pp 743–750. doi:10.1109/ICWS.2007.91
13. Wang XZ, Wang ZJ, Xu XF, Liu Y, Chu DH (2010) A service composition approach for the fulfillment of temporally sequential requirements. In: Proceedings of IEEE congress on service, July 2010, pp 559–565
14. Wang XZ, Wang ZJ, Xu XF, Liu Y (2011) A service composition method for tradeoff between satisfactions of multiple requirements. J Comput Res Dev 48(4):627–637
15. Bouwman H, Haaker T, de Vos H (2007) Mobile service bundles: the example of navigation services. Electron Mark 17(1):20–28. doi:10.1080/10196780601136757
16. Dhanesha KA, Hartman A, Jain AN (2009) A model for designing generic services. In: Proceedings of 2009 IEEE international conference on services computing, Sept 2009, pp 435–442. doi:10.1109/SCC.2009.75
17. Blau B, Block C, Stöße J (2008) How to trade electronic services: current status and open questions. In: Group decision and negotiation (GDN) meeting, June 2008
18. Chang W-L, Yuan S-T (2009) A Markov-based collaborative pricing system for information goods bundling. Expert Syst Appl 36(2):1660–1674. doi:10.1016/j.eswa.2007.11.040
19. Akkermans H, Baida Z, Gordijn J, Pena N, Altuna A, Laresgoiti I (2004) Value webs: using ontologies to bundle real-world services. IEEE Intell Syst 19(4):57–66. doi:10.1109/MIS.2004.35
20. Kohlborn T, Luebeck C, Korthaus A, Fielt E, Rosemann M, Riedl C, Krcmar H (2010) Conceptualizing a bottom-up approach to service bundling. Adv Inf Syst Eng Lect Notes Comput Sci 6051/2010:129–134. doi:10.1007/978-3-642-13094-6_11
21. Jones A, Rabelo LC, Sharawi AT (1999) Survey of job shop scheduling techniques, Wiley Encyclopedia of Electrical and Electronics Engineering. Wiley, New York. doi:10.1002/047134608X.W3352
22. Dong F, Akl SG (2006) Scheduling algorithms for grid computing: state of the art and open problems. School of Computing, Queen's University, Kingston, technical report No. 2006-504
23. Czajkowski K (2006) Globus GRAM. Globus Toolkit Developer's Forum, Globus Alliance
24. Henderson RL (1995) Job scheduling under the portable batch system. Lect Notes Comput Sci 949/1995:279–294. doi:10.1007/3-540-60153-8_34
25. Bayucan A, Henderson RL, Jones JP et al (2000) Portable batch system (OpenPBS Release 2.3) administraotor guide. Veridian Corporation, Aug 2000. http://ladon.iqfr.csic.es/docs/openpbs_v2.3_admin.pdf
26. Liu P, Raahemi B, Benyoucef M (2011) Knowledge sharing in dynamic virtual enterprises: a socio-technological perspective. Knowl Based Syst 24(3):427–443. doi:10.1016/j.knosys.2010.12.004
27. Ulungu EL, Teghem J (1993) Multi-objective combinatorial optimization problems: a survey. J Multi Criter Decis Anal 3(2), Oct 1993. doi:10.1002/mcda.4020030204