



# MatchCom: Stable Matching-Based Software Services Composition in Cloud Computing Environments

Satish Kumar<sup>1</sup>✉, Renyu Yang<sup>2</sup>, Rajiv Ranjan Singh<sup>3</sup>, Rami Bahsoon<sup>4</sup>, Jie Xu<sup>5</sup>, and Rajkumar Buyya<sup>6</sup>

<sup>1</sup> School of Built Environment, Engineering and Computing, Leeds Beckett University, Leeds, UK

s.kumar@leedsbeckett.ac.uk

<sup>2</sup> School of Software, Beihang University, Beijing, China

renyu.yang@buaa.edu.cn

<sup>3</sup> Department of Cyber Security and Networks, Glasgow Caledonian University, Glasgow, UK

rajiv.singh@gcu.ac.uk

<sup>4</sup> School of Computer Science, University of Birmingham, Birmingham, UK

r.bahsoon@cs.bham.ac.uk

<sup>5</sup> School of Computing, University of Leeds, Leeds, UK

j.xu@leeds.ac.uk

<sup>6</sup> School of Computing and Information Systems, University of Melbourne, Melbourne, Australia

rbuyya@unimelb.edu.au

**Abstract.** User preferences on throughput, latency, cost, service location, etc. indicate specific requirements when choosing a web service from the cloud marketplace. Service providers can also adopt preferences to prioritize a set of end-users based on their Service Level Agreement and service usage history. An effective matching between preferences from both parties enables fair service marketing in the cloud marketplace. The existing approaches are insufficient in capturing both parties' preferences in the service composition process. To address this limitation, we propose MatchCom, a novel service composition approach driven by diverse preferences and formulate it as the stable marriage problem. Particularly, we present a novel fair preference ordering mechanism – in the context of a cloud marketplace, for enabling users to specify services provider ranking based on the capability they can provision, and for helping providers select the most suitable users to be served given users' profile. MatchCom extends the Gale-Shapely Algorithm with a service composer algorithm for optimising the stable service composition. We evaluate MatchCom on a service-oriented system with 10 abstract services, each of which has 100 candidate web services. We establish through the experimental results that MatchCom outperforms other baseline approaches and can maximize end-user satisfaction in the composition process.

**Keywords:** Service Composition · Quality of Services · Stable Matching

## 1 Introduction

The shift of industrial IT services to cloud-based service models has made service composition a key driving force for building on-demand service-oriented software appli-

cations by composing multiple existing web services from the cloud marketplace [1]. However, the emergence of multiple functionally equivalent web services with different Quality of Service (QoS) values in the cloud marketplace can present challenges when selecting an optimal web service for composing software applications [2]. Further, it can be challenging when multiple users express different preferences and constraints to get the same service in the cloud marketplace. On the other hand, cloud service providers aim to maintain a positive service reputation while maximizing service revenue [2,3]. To achieve this, it is important to prioritize users who have long-term business potential based on their Service Level Agreements (SLAs), service usage or other factors. In this context, preferences could be an effective mechanism for creating fair marketing in the cloud marketplace. Users express their preferences on service QoS constraints, service location, service cost, and reputation; and service providers rank the users based on their SLAs and service matches. However, existing research studies [4–6] have the limitation of supporting only end-user preferences, neglecting service provider’s preferences when provisioning suitable web services.

To address these challenges, we present a novel Stable Matching Based Service Composition called *MatchCom* that explicitly captures the end user’s and service provider’s preferences and optimizes the preference stability-aware service composition. We employ Stable Marriage Problem [7] to model our service composition approach. Our key idea is to capture the end-user’s preferences from their SLAs. Further, we use these preferences to rank all similar functionally equivalent web services that exhibit different QoS values. Similarly, the service provider ranks end users based on their SLA types. Then, we apply GSA-based *MatchCom* to find the stable service matches to form the composition solutions.

In a nutshell, the major contributions of this paper are ① we formulate the service composition as a stable marriage problem. ② we model a preference generation scheme for both end users and service providers in the global cloud marketplace. This scheme facilitates finding stable service matches driven by the service provider’s and end-user’s preferences over each other. ③ we tailor a Gale-Shapley Algorithm (GSA) and present a *serviceComposer* algorithm that tends to maximize the end-user’s satisfaction in the composition process. ④ we evaluate *MatchCom* on a service composition system with up to 10 abstract services workflow, each of which has 80 to 100 candidate web services, under different QoS values derived from the real-world WS-DREAM dataset [9].

## 2 Stable Matching Based Service Composition

The stable marriage problem, introduced by Gale and Shapley in 1962 [7], involves matching two sets of agents, such as men and women. A crucial aspect of this problem is the ordering of preferences, where each man and woman rank each other in a strict order of preference and then a Gale and Shapley algorithm exploits these preferences to generate stable matches. We leverage this approach in our research with a particular focus on the diverse preferences-based service compositions in the global cloud marketplace; where QoS constraints, service budget, service region, and SLA types are considered the most preferred parameters for both entities to establish a strict order of preference over each other.

## 2.1 Preference Order Modelling

Here, we assume that there are  $x$  number of users  $U_i, i = (1, 2, 3, \dots, x)$  and each user requests  $y$  number of tasks in the composite software application  $U_{ij}, j = (1, 2, 3, \dots, y)$  with  $z$  dimension service constraints (e.g., QoS, Cost, service region/location  $S^{loc}$ )  $U_{ijk}, k = (1, 2, 3, \dots, z)$  for choosing a web service in the composition. On the other side, based on functional (e.g., task) and QoS requirements, the service providers offer  $m$  set of web services  $S_p, p = (1, 2, 3, \dots, m)$  in the global cloud marketplace and each set contains  $n$  candidate web services that are functionally equivalent to perform  $j^{th}$  task of user  $U_{ij}, S_{pq}, q = (1, 2, 3, \dots, n)$  and each candidate web service has  $l$  dimension QoS values  $S_{pqr}, r = (1, 2, 3, \dots, l)$  and SLAs type provisioned by cloud service providers as part of service delivery. Therefore, a matching model is defined  $M(U, S)$ , where each user  $U_i$  needs to rank all candidate web services in a set  $S_{pq}$  for a  $j^{th}$  task that satisfies the service location constraint  $\mathbb{CL}$ ; otherwise, underlying candidate web service will be discarded from the ranking process using Eq. 1.

$$(S_{pq}) = \begin{cases} 1 & \text{if } \mathbb{CL} = S_{pq}^{loc} \\ 0 & \text{if otherwise} \end{cases} \quad (1)$$

Similarly, the service provider ranks all users  $U_i$  based on their SLAs type and service QoS values  $S_{pqr}$  legally provisioned in the SLA [2]. Therefore, the preference of  $j^{th}$  task of a user  $U_i$  over the  $q^{th}$  candidate web service  $S_{pq}$  is computed by aggregating the preference of each QoS constraint over QoS value exhibited by the candidate web service or vice-versa [4]. Further, we compute the best-case and worst-case values of each QoS objective (e.g., QoS constraint) imposed by a user or offered by the service providers as part of their service delivery. In the case of positive QoS (e.g., throughput) constraints criteria  $\mathbb{CQ}^+$ , the best case indicates the expected  $r^{th}$  QoS value of a web service must be larger than or equal to the required constraint value of  $k^{th}$  objective (constraint weight) of a user  $U_i$ , otherwise expected objective value consider as the worst-case value for the  $k^{th}$  constraint of a user  $U_i$  [6]. However, we calculate the exact values for the best-case and worst-case of each QoS objective required by the users over candidate web services. Further, the best-case and worst-case values are multiplied by +1 and -1, respectively, which shows how much the expected value is good or bad for each required objective of the user, as shown in Eq. 2.

$$\mathbb{CQ}^+(S_{pq}) = \begin{cases} \frac{U_{ijk}}{S_{pqr}} \times (-1) & \text{if } S_{pqr} < U_{ijk} \\ \frac{S_{pqr}}{U_{ijk}} \times (+1) & \text{if } S_{pqr} \geq U_{ijk} \end{cases} \quad (2)$$

$$\mathbb{CQ}^-(S_{pq}) = \begin{cases} \frac{S_{pqr}}{U_{ijk}} \times (-1) & \text{if } S_{pqr} > U_{ijk} \\ \frac{U_{ijk}}{S_{pqr}} \times (+1) & \text{if } S_{pqr} \leq U_{ijk} \end{cases} \quad (3)$$

where  $k$  indicates the  $z^{th}$  constraint weight for the  $j^{th}$  task of user  $U_i$ .

Similarly, for the negative QoS (e.g., response time) constraints criteria  $\mathbb{CQ}^-$ , best-case shows the expected objective value should be smaller than the required objective

value of a user  $U_i$ , other than shows the worst-case value. These values are calculated using Eq. 3.

We generate  $i^{\text{th}}$  user preference  $\mathbb{P}(U_{ij})$  for the  $j^{\text{th}}$  task over candidate web services by computing the net ranking value using Eq. 4.

$$\mathbb{P}(U_{ijk}) = \sum_{r=1}^l \mathbb{C}Q^+(S_{pqr}) + \sum_{r=1}^l \mathbb{C}Q^-(S_{pqr}) \quad (4)$$

$$\mathbb{P}_{sla}(U_i) = \begin{cases} w_s & \text{if } U_i = (sla = Silver) \\ w_g & \text{if } U_i = (sla = Gold) \\ w_p & \text{if } U_i = (sla = Platinum) \end{cases} \quad (5)$$

---

**Algorithm 1:** matchGenerator( $U, PU, PS$ )
 

---

```

1 Input: The set of users  $U$  and service provider's web services  $S$ . Users preferences  $PU$ 
  ( $\forall u_{ij} \in PU$ ) and candidate web services  $PS$  ( $\forall s_{pq} \in PS$ )
2 Output: array matrix  $C$ 
3 Initialization:  $\forall u_{ij} \in PU$  and  $\forall s_{pq} \in PS$  to be free,  $M \leftarrow \emptyset$ 
4 for  $\forall u_{ij} \in U$  do
5   while  $s_{pq} \in PS$  is free and  $PS \neq \emptyset$  do
6      $u_{ij} = j^{\text{th}}$  task of user  $u_i$  highest ranked on  $q^{\text{th}}$  web service of set  $s_p$  to whom  $q^{\text{th}}$ 
      has not proposed yet
7     if  $u_{ij}$  is free then
8       assign  $q^{\text{th}}$  web service to  $j^{\text{th}}$  task of  $u_i$ 
9        $M \leftarrow M \cup (u_{ij}, s_{pq})$ 
10    end
11    else if ( $u_{ij}$  prefers  $q^{\text{th}}$  web service over previous assigned  $n^{\text{th}}$  web service of set
       $s_p$ ) then
12      assign  $q^{\text{th}}$  web service to  $j^{\text{th}}$  task of  $u_i$ 
13       $M \leftarrow M \cup (u_{ij}, s_{pq})$ 
14      assigned  $n^{\text{th}}$  web service to be free  $M \leftarrow M \setminus (u_{ij}, s_{pn})$ 
15    end
16    else
17       $j^{\text{th}}$  task of  $u_{ij}$  rejects  $q^{\text{th}}$  web service of set  $s_{pq}$  (and  $q^{\text{th}}$  remain free)
18    end
19  end
20   $C \leftarrow M$ 
21 end
22 serviceComposer( $C, U$ )

```

---

$$\mathbb{P}(S_{pqr}) = [\sum_{k=1}^z \mathbb{CQ}^+(U_{ijk}) + \sum_{k=1}^z \mathbb{CQ}^-(U_{ijk})] \times \mathbb{P}_{sta}(U_i) \quad (6)$$

On the other hand, we consider diverse types of SLAs offered by cloud service providers as part of web service delivery in the cloud marketplace. Suppose end-users negotiated different types of SLAs such as silver, gold and platinum with the cloud service providers [11]. In this respect, the cloud service provider gives the highest priority to a user who has platinum SLA rather than gold and silver users SLA, as shown in Eq. 5. For the sake of simplicity, in this work, we give some weight to distinguish each SLA says silver ( $w_s = 0.1$ ), gold ( $w_g = 0.3$ ) and platinum ( $w_p = 0.5$ ). The cloud service provider gives preference over the users  $U_i$  based on their SLA types and service QoS provision documented in their SLA. We compute the net ranking over each user service demand using Eq. 6.

## 2.2 Software Services Composition

After generating the preference matrices  $PU$  and  $PS$  using Eq. (4) and (6) respectively, Algorithm 1 shows the process of applying the Gale-Shapley Algorithm (GSA) to find the optimal stable match  $M(j, q)$  between the  $j^{th}$  task of user  $u_i$  and  $q^{th}$  web service of set  $s_p$ . The preference matrices of all users  $PU$  and service provider's web services  $PS$  are provided as input to the algorithm and initialize all parameters that will be used in the next phase (Lines 5–17). From Lines 5–9, the user  $u_i$  selects the most preferred  $q^{th}$  web service from the candidate web services set  $s_p$  and form the matching  $M(u_{ij}, s_{pq})$  if it is not matched with other tasks of user  $u_i$  in the  $PU$  list. From Lines 10–13, if a  $j^{th}$  task of user  $u_i$  has already had the match of  $n^{th}$  candidate web service in the  $PS$  list but  $j^{th}$  task prefers to  $q^{th}$  web service in the  $PS$  list over the current  $n^{th}$  web service

---

### Algorithm 2: serviceComposer( $C, U$ )

---

```

1 Input:  $C$  and  $U$ 
2 Output: service composition matrix  $u_i(cs)$ 
3 Initialization:  $u_i(cs) \leftarrow \emptyset, u_d \leftarrow \emptyset$ 
4 while  $U \neq \emptyset$  do
5   for  $\forall u_i \in U$  do
6      $C_{global} \leftarrow checkQoS(u_i, C_i)$ 
7     if ( $C_{global}$  satisfy user  $u_i$  constraints) then
8        $u_i(cs) \leftarrow C_i$ 
9     else
10       $u_d \leftarrow C_i$ 
11    end
12  end
13   $U \leftarrow u_d$ 
14  matchGenerator( $U$ )
15 end
16 return  $u_i(cs)$ 

```

---

match. Then, a new optimal match  $M(u_{ij}, s_{pq})$  is formed and further, makes the  $n^{th}$  web service free in the  $PS$  list of candidate web services. However, the  $j^{th}$  task of user  $u_i$  rejects the  $q^{th}$  web service request in the matching process if it already had the higher preference ranking web service match than the preference ranking of  $q^{th}$  web service (Lines 14–16). From Lines 5–17, this process is repeated until the first task of all users  $u_i$  in  $PU$  list assigned the optimal web service from the  $PS$  list. After completing the first iteration, the  $q^{th}$  web service assigned to the first  $j^{th}$  task of all users  $U$  are stored in the array matrix  $C$  (line 19), and this step is repeated until all users' tasks assigned the set of optimal web services (Lines 4–19).

However, Algorithm 1 produced the sets of concrete web services to form the composition solutions for all the users requesting services in the cloud marketplace. Further,  $C$  is provided as input to Algorithm 2 for performing a next-level composition process that guarantees to satisfy all constraints imposed by end-users. In line 6, we calculate the aggregated QoS values and cost of all web services in  $C_i$  for the user  $u_i$  using QoS aggregation methods [11] and then check whether the global constraints are satisfied or not imposed by  $i^{th}$  user  $u_i$  (line 7). If true, then form the composite service for the user  $u_i$  (line 8), otherwise, user  $u_i$  rejects the composition plan and demands a new service composition plan, such user IDs are recorded in array  $u_d$ . This process is repeated until all web service sets in  $C$  are checked (4–11). In line 13, the current users set  $U$  is updated with users set  $u_d$  who demand the new service composition plans over the current infeasible plan. Further, Algorithm 1 is invoked with updated user set  $U$  to find the optimal set of web services for the users  $u_d$ . The whole process is repeated until users set  $U$  to get empty and then return the optimal service compositions plans  $u_i(c.s)$  for all the users  $u_i$ .

### 3 Performance Evaluation

Our experiments aim to answer the research questions – **RQ1:** Is MatchCom approach more stable than the baseline approach?; **RQ2:** How MatchCom can outperform other baselines including evolutionary algorithm-based approaches?; **RQ3:** What is the running overhead of MatchCom compared to other approaches?

#### 3.1 Experiment Setup

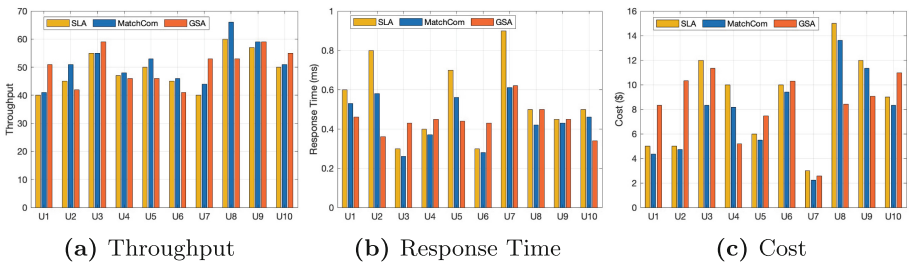
For experiment purposes, we employed a service composition system with 10 abstract services, which are sequentially connected to construct a service composition workflow [8]. Further, we deployed 100 candidate web services to perform each abstract service in the composition workflow. However, each candidate web service exhibits different QoS values, which are randomly picked from the real-world WSDream dataset [9]. Further, the service cost value and service region are generated randomly for each candidate web service participating in the composition. Apart from that, we randomly create end-user service requirements (number of tasks in the service, throughput, response time, service cost, and service region), which are generally documented in the end user's SLA.

### 3.2 Results and Discussion

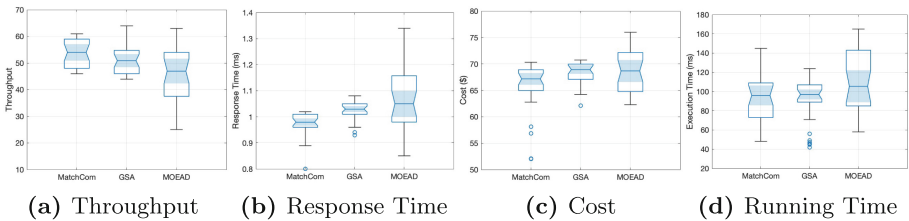
To answer the above RQs, we examine the performance of MatchCom against Baseline (GSA) [10] and MOEAD [11] based approaches.

**RQ1: SLAs Stability of MatchCom Against Baseline:** To answer RQ1, we plot the service composition plans optimized by MatchCom and GSA approaches as shown in Fig. 1. In particular, we examine the end-users SLA constraints such as throughput, response time and cost on whether the composed service compositions are satisfied or not. As can be seen from Fig. 1a and 1b, SLA throughput and response time constraints are violated by the composition solutions generated by GSA approach for the end-users  $u_4$ , and  $u_6$ , whereas, MatchCom satisfies all users SLA constraints. Further, an interesting insight is shown from a cost perspective as shown in Fig. 1c, GSA satisfies the QoS constraints for the end-user  $u_{10}$  but fails to meet the service budget requirement. However, MatchCom does not guarantee the higher values of QoS constraints satisfaction but satisfies all constraints under given budget requirements.

**RQ2: Performance of MatchCom:** To investigate RQ2, we assess the performance of MatchCom by comparing with baseline and evolutionary algorithm (MOEAD) based approaches. We run all approaches 30 times and record the best QoS value of throughput, response time and cost objectives from the optimal set of service composition solutions generated in each run. As shown in the boxplots of Fig. 2a and Fig. 2b, we see that MatchCom achieve much better QoS values for throughput and response time objectives with small variance than GSA and MOEAD approaches. Also, GSA obtains better QoS



**Fig. 1.** Users SLA constraints achieved by MatchCom and GSA approaches (users = 10, workflow: number of tasks = 10, number of candidate web service for each task = 100).



**Fig. 2.** Throughput, Response Time, Cost and Running Time yield by MatchCom, GSA and MOEAD approaches.

objectives values than MOEAD. Further, as we can see from Fig. 2c, overall MatchCom achieves better QoS values with less cost than GSA and MOEAD. Overall, MatchCom outperforms other approaches in achieving a better QoS value for each objective in the composition.

**RQ3: Running Time of MatchCom:** To understand RQ3, we plot the running time of all approaches as shown in Fig. 2d. As we can see MOEAD is the slowest due to exploiting a huge search space of  $X^N$  ( $X$  denotes an abstract service, and  $N = 100$  is the number of candidate services to perform  $X$  abstract service). However, GSA and MatchCom take less execution time than MOEAD because they reduce the search space by discarding all candidate web services they are unable to satisfy the service region constraints mentioned in the end-user's preferences (constraints). But, MatchCom is slower than GSA because it favors maximising the end-user satisfaction in the composition process whereas GSA does not care to satisfy all user's constraints, as we have shown in answering RQ1 and RQ2.

## 4 Conclusions

In this paper, we proposed a stable matching-based service composition approach called MatchCom leveraging stable marriage problem. We introduced a novel bilateral preference model that gives equal ownership to service providers and end users for fairly serving and consuming services in cloud marketplace. MatchCom service composer can generate fair preference ordering for both service providers and end users. The GSA produces stable service matches which the built-in service composer further uses to optimize the service composition solutions. Experimental results show that MatchCom is more effective than baseline approaches and favors to maximize the end-user's satisfaction in the composition.

## References

1. Bi, X., Yu, D., Liu, J., Hu, Y.: A preference-based multi-objective algorithm for optimal service composition selection in cloud manufacturing. *Int. J. Comput. Integr. Manuf.* **33**(8), 751–768 (2020)
2. Kumar, S., Chen, T., Bahsoon, R., Buyya, R.: DebtCom: technical debt-aware service recomposition in SaaS cloud. *IEEE Trans. Serv. Comput.* **16**(4), 2545–2558 (2023)
3. Pudasaini, D., Ding, C.: Service selection in a cloud marketplace: a multi-perspective solution. In: 2017 IEEE 10th International Conference on Cloud Computing (Cloud), pp. 576–583 IEEE (2017)
4. Wang, H., Ma, P., Yu, Q., Yang, D., Li, J., Fei, H.: Combining quantitative constraints with qualitative preferences for effective non-functional properties-aware service composition. *J. Parallel Distrib. Comput.* **100**, 71–84 (2017)
5. Choi, C.R., Jeong, H.Y.: A broker-based quality evaluation system for service selection according to the QoS preferences of users. *Info. Sci.* **77**, 553–566 (2014)
6. Wang, H., Chiu, W., Wu, S.C.: QoS-driven selection of web service considering group preference. *Comput. Netw.* **99**(1), 111–124 (2015)
7. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. *Am. Math. Monthly.* **69**(1), 9–15 (1962)



8. Kumar, S., Chen, T., Bahsoon, R., Buyya, R.: DATESSO: self-adapting service composition with debt-aware two levels constraint reasoning. In: 2020 IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-managing Systems, pp. 96–107. IEEE/ACM (2020)
9. Zheng, Z., Zhang, Y., Lyu, M.R.: Investigating QoS of real-world web services. *IEEE Trans. Serv. Comput.* **7**(1), 32–39 (2012)
10. Li, F., Zhang, L., Liu, Y., Laili, Y.: QoS-aware service composition in cloud manufacturing: a Gale-Shapley algorithm-based approach. *IEEE Trans. Syst. Man Cyber. Syst.* **50**(7), 2386–2396 (2020)
11. Kumar, S., Chen, T., Bahsoon, R., Buyya, R.: Multi-tenant cloud service composition using evolutionary optimization. In: 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), pp. 972–979. IEEE (2020)