Multi-Server Stable Rendezvous for the Metaverse

Ningxin Su, *Student Member, IEEE,* Baochun Li, *Fellow, IEEE,* Bo Li, *Fellow, IEEE*

Abstract—When the real world and the digital world meet, they create a shared virtual space called the *metaverse*, involving multiple virtual communities in which users interact with one another in a highly immersive and interactive fashion. Due to the pressing need for scalability when handling millions of users in the metaverse regardless of the applications and services provided, it is imperative to deploy multiple servers across geographically distributed datacenters around the world. In this paper, we envision, design, and implement a new rendezvous service between a large number of users and a collection of geographically distributed servers, taking into account the latencies and pair-wise network bandwidth between the users and the servers, as well as bandwidth and processing capacity constraints on the servers themselves when handling the users. Our new rendezvous service is designed with simplicity and efficiency as its primary objective, and uses a revised design of the *Deferred Acceptance* algorithm to guarantee a stable matching between the users in the metaverse and its servers. As a case study, our rendezvous service has been implemented in the context of a federated learning application.

1 Introduction

The metaverse is a shared virtual place generated by the confluence of the physical and digital universes. People, organizations, and information can interact in a highly immersive, interactive, and multi-dimensional environment. Due to recent technological advances in virtual and augmented reality [1], a notion that has been pondered in science fiction for decades is now becoming a reality. Users can create and design their own avatars to communicate with others, explore virtual locations, and engage with one another in a wide variety of applications [2]. With such an engaging virtual environment, the metaverse has the potential to facilitate new *communities* in which large numbers of users participate to play, work, socialize, educate, and to explore mutual interests in general in a highly engaging fashion. These communities may include virtual concerts, sports events, games, business meetings, and classrooms, as well as any other forms of interactive storytelling.

One of the most fundamental characteristics of the metaverse is the sheer *scale* of its concurrently active users. Even in their infancy, current virtual worlds, such as Roblox and Fortnite, are already used by hundreds of millions of users.[1] As has been well recognized in the context of web services, such a large number of users in the metaverse can only be adequately served with a collection of servers that are hosted at geographically distributed datacenters around the world.

With such a pressing need for scalability in the metaverse, as well as the ensuing demand for deploying a collection of geographically distributed servers, a practical technical challenge naturally arises: Which server should be used to serve a particular user, and more generally, how should a large number of users be matched to a small number of servers serving the metaverse? In conventional web services, a web client typically chooses an edge Content Distribution Network (CDN) server that is the closest with respect to rather crude estimates of network latencies using the Domain Name Service and the client's IP address. Though such a simple solution is fully decentralized and more scalable, it does not consider the processing and bandwidth capacities on the servers, pair-wise network link bandwidth between clients and the servers, as well as potentially more accurate real-time estimates of network latencies.

In the context of the metaverse, however, it has been commonly accepted that network latencies and available bandwidth between a user and its server are much more crucial than conventional web services. In fact, latencies and achievable throughput are so important that user experiences with some metaverse applications may be severely downgraded to the point that they may not be functional. Such emphasis on network latencies and achievable throughput motivates a more metaverse-centric design, where user-server matching decisions are produced by a new *rendezvous* service, using up-to-date, and therefore more accurate, estimates of pair-wise latencies and available bandwidth between metaverse users and their servers.

In this paper, we propose a new scalable rendezvous service for matching a large number of users to their corresponding servers. Our research objective is to design a simple, decentralized, and scalable service that can be readily implemented and deployed in practice. The initial, and quite intuitive, solution for such a matching problem is to formulate it as an optimization problem that incorporates both the capacity constraints on the servers and the optimization objectives involving network latencies and link bandwidth between metaverse users and their servers. However, solving such an optimization problem, even with just an off-the-shelf linear solver, would require a centralized implementation that requires collecting all link-level network measurements at a central server, which is far from scalable in practice. Instead, we argue that, in the metaverse, such a matching problem should be solved using a fully decentralized solution that takes into

Ningxin Su and Baochun Li are with the Department of Electrical and Computer Engineering, University of Toronto. Their email addresses are ningxin.su@mail.utoronto.ca *and* bli@ece.toronto.edu*, respectively.*
Bo Li is with the Department of Computer Science, Hong Kong University of Science and Technology. His email address is bli@cse.ust.hk.

1. Key metaverse statistics are available online at https://www.luisazhou.com/blog/metaverse-statistics/.

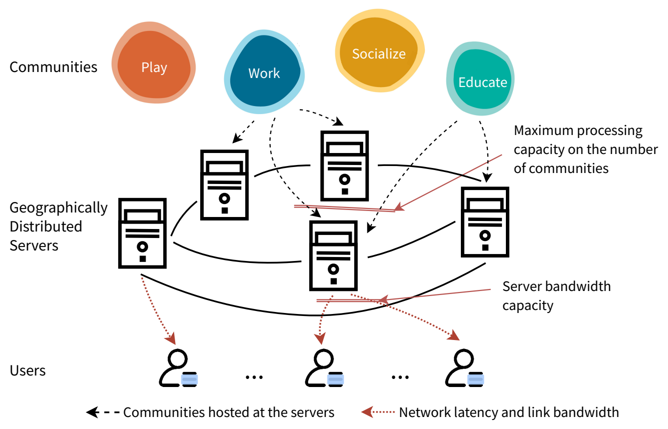

Communities
Play
Work
Socialize
Educate
Maximum processing capacity on the number of communities
Geographically Distributed Servers
Server bandwidth capacity
Users
...
...
Communities hosted at the servers
Network latency and link bandwidth

Fig. 1. A variety of preferences involving network latencies and link bandwidth, as well as constraints involving bandwidth and processing capacities at the servers, need to be considered when designing a new rendezvous service of matching users to geographically distributed servers around the world.

account community interests of the metaverse users, capacities on the servers, as well as link latencies and bandwidth between the users and the servers. Such a decentralized solution should also be sufficiently general and applicable to a wide variety of applications in the metaverse.

To achieve such an objective, we propose to use a revised design of the *deferred acceptance* algorithm to produce a *stable matching* between the metaverse users and their servers, without violating the server capacity constraints. Our proposed algorithm is both simple and decentralized in nature, and enjoys a polynomial-time complexity just as the conventional deferred acceptance algorithm for the college admissions problem [3]. Due to the importance of link-level pair-wise network latencies and bandwidth between the users and the servers in the metaverse, our algorithm prioritizes them over resource utilization on the servers.

Our original contributions in this paper are three-fold. *First*, we propose a new *rendezvous* service between a large number of metaverse users and their servers to improve the scalability of metaverse applications, especially when multiple virtual communities need to be simultaneously supported, each consisting of its own group of users. *Second*, given the significance of network latencies and available bandwidth in metaverse-centric applications, we propose a scalable and simple user-server matching algorithm based on deferred acceptance, which can be implemented in a fully decentralized fashion, and show that the matching it produces is stable. *Finally*, We demonstrate the performance of our algorithm by training globally shared machine learning models using federated learning (FL) [4], a privacy-preserving distributed training mechanism, as a case study. In our experiments, we simulate the communication between users and servers during FL sessions and varying network environments in the metaverse, using the MNIST and CIFAR-10 datasets for image classification tasks. Our experimental results show that, in cases of large-scale concurrent users and dynamic network environments, periodically running our rendezvous matching algorithm has minimal impact on our tasks due to its simplicity.

2 Preliminaries and Related Work

Scalability. Constructing large-scale network topologies that connect users to a number of servers in geographically distributed datacenters is required for the sake of better scalability, which has been widely recognized as one of the fundamental challenges in the metaverse [1], [5]. This is particularly the case for the metaverse, where millions of users are expected to interact with one another within the confines of virtual communities. At the same time, metaverse applications have exacting requirements for ultra-high network bandwidth and ultra-low network latencies. While discussions of the specifics of how a real-world metaverse network may be engineered to meet these requirements — with better congestion control algorithms or better utilization of the wireless spectrum with beyond 5G technologies, for example — it is worth noting that both available bandwidth and network latencies in real-world metaverse networks tend to vary over time in practice. In this paper, we focus on designing a decentralized algorithm to construct the network topology between metaverse users and their servers, with the reasonable assumption that network latencies and available bandwidth can be readily measured (or predicted using historical estimates) between each individual user and all the servers in the metaverse.

Metaverse communities. The concept of having virtual communities in the metaverse is quite intuitive: consider a virtual 3D world with multiple gathering locations, both indoors and outdoors, that a user can roam to and start interacting with other users at the same virtual location. Existing literature on the metaverse has also made it clear that metaverse communities are important for self-governance, forming an autonomous Code of Conduct that can be enforced using blockchain technologies [5]. A community can also take the form of a *sub-metaverse* [5], each of which can offer various kinds of goods and services, such as gaming, social dating, online venues such as museums, and online events such as concerts. In this paper, the term *community* has general connotations that can represent either sub-metaverses [5], virtual locations in a 3D world [1], or virtual environments.

On the one hand, at any given moment, it is reasonable to assume that a user may only be engaged in one activity in the metaverse — such as working or playing a game — which implies that each user can only participate in one community at any given time. On the other hand, as we are deploying a small number of servers in geographically distributed datacenters to serve a large number of users, it is straightforward to observe that each server has a maximum *bandwidth capacity* when serving their users in the metaverse. In addition, though it is natural to conceive that each server can serve multiple communities in the metaverse, due to hardware resource constraints such as CPU, memory, and storage constraints, we assume that there exists an upper bound with respect to the number of communities that can be maximally served by each server, which we henceforth refer to as its *processing capacity*.

Stable matching. The fundamental problem of resource allocation, which existed prevalently in cloud computing, networking, and wireless communications, ranging from channel assignment to job scheduling, has typically been formulated as a combinatorial optimization problem (Chieochan *et al.* [6], for example, formulated channel assignment and coding as a joint optimization problem). In our problem in the context of the metaverse, we are also interested in assigning metaverse users to their servers in a metaverse network with the presence of server capacity constraints, which can also be formulated as an optimization problem with the objective of optimizing the utilization of resources.

However, we argue that solving optimization problems using off-the-shelf optimization solvers is centralized, and may not be scalable to millions of users. In this paper, we advocate the use of *stable matching* [3] instead, which enjoys a number of salient advantages. *First*, stable matching is a general framework that uses *preferences* to model a user's (or a server's) interests, and *stability* as a solution concept rather than *optimality*. As a solution framework for general matching problems in networking [7], it is well suited for our problem at hand. *Second*, the deferred acceptance algorithm, designed to solve the stable marriage and college admissions problems [3], is general, simple, and efficient with a polynomial-time complexity, and can be fully decentralized. *Finally*, stable matchings are known to be in the core of the market that cannot be improved upon by a coalition of agents [8].

In the literature, due to its efficiency and simplicity of implementation, stable matching has been adopted to solve several cloud computing, networking, and wireless communication problems (*e.g.*, [7], [9]). In this paper, we will focus on formulating our problem of architecting a new decentralized *rendezvous* service in the context of stable matching theory, and proposing a revised design of the deferred acceptance algorithm that guarantees the stability of the matching outcomes between metaverse users and their servers, which we later refer to as a *stable rendezvous*. There does not exist any previous work in the literature that addressed the same challenge.

3 Stable Rendezvous: Algorithm Design

We are now ready to design a new decentralized algorithm for *stable rendezvous* using the stable matching theory and a revised version of the deferred acceptance algorithm.

3.1 Stable Matching as the Solution Concept

As we discussed in Section 2, the dearth of existing work in the literature that addresses the challenge of matching a large number of metaverse users to a small number of geographically distributed servers allows us to start with a clean-slate design.

One design objective of the utmost importance is that the algorithm needs to be *fully decentralized*. This objective, though seemingly obvious due to our emphasis on *scalability*, motivates the use of stable matching theory, and rules out two alternatives that are widely used in the cloud computing and networking literature to solve resource allocation problems in general: *First*, optimization-based solutions formulated the problem at hand as a combinatorial optimization problem and used off-the-shelf solvers or heuristics to solve it (*e.g.*, [6]). Both heuristic algorithms and off-the-shelf optimization solvers are inherently centralized, requiring the input to be collected beforehand. Centralized solutions are not scalable and cannot be executed asynchronously in a distributed fashion. *Second*, game-theoretic approaches, such as algorithms based on the concept of Nash Bargaining Solutions (*e.g.*, [10]) or double auctions (*e.g.*, [11]), were also centralized in nature, and some heuristics, such as those designed to compute Nash Bargaining Solutions, were computationally hard to compute as well.

Stable matching theory, on the other hand, offers a different solution concept compared to optimization and game theory. Rather than optimizing for a particular objective function or computing the Nash equilibrium, we seek to achieve *stability* instead. The merit of the stable matching framework, as shown in related work in the literature [7], is its overall practicality due to its decentralized algorithmic design. Depending on the nature of the problem at hand, it is likely that the deferred acceptance algorithm [3] can be applied with a revised design.

To introduce the basic concept of stability, we start by introducing a classic stable matching problem that is the closest to our problem at hand of matching metaverse users to their servers: the *college admissions* problem [3].

In the college admissions problem, a set of n applicants, $\mathcal{A}$, is to be assigned among m colleges $\mathcal{C}$, each of which has a quota q_i of applicants it can maximally accommodate. Each applicant ranks the colleges in the order of his/her preference, and each college similarly ranks applicants who have applied for it in order of its preference. For convenience we assume there are no ties in the rankings. Each ranking, also called a *preference*, at both applicants and colleges is denoted by $\succ$. For example, $a \succ_i b$ in an applicant i's ranking implies that the applicant prefers to attend college a rather than b. If the applicant i prefers to remain unmatched rather than being matched with college c (i.e., $\emptyset \succ_i c$), then c is considered unacceptable to the applicant.

The outcome of an algorithm solving the college admissions problem is a *matching* between applicants and colleges, defined in general as:

Definition 1 (Matching)**.** *An outcome of the college admissions problem is a* matching $\mu : \mathcal{A} \times \mathcal{C} \to \mathcal{A} \times \mathcal{C}$ *such that* $a \in \mu(c)$ *if and only if* $\mu(a) = c$, *and* $\mu(a) \in \mathcal{C} \cup \emptyset, \mu(c) \subseteq \mathcal{A} \cup \emptyset, \forall c, a$.

This implies that an applicant can be unmatched to any college—and a college can also be unmatched to any applicant—but when an applicant a is matched to a college c, the college's matched list of applicants must include this applicant a. In other words, a matching, as an outcome, matches applicants on one side to colleges on the other side, or to the empty set.

We now need further qualifications to distill a desirable set of matchings from all possible outcomes. Naturally, a *stable* matching is desirable, and such *stability* is defined using the concepts of individual rationality and blocking pairs:

Definition 2 (Individual rationality)**.** *A matching is* individual rational *if and only if there does not exist an applicant* a *(or a college* c*) who prefers being unmatched to being matched with* $\mu(a)$ *(or* $\mu(c)$*), i.e.* $\emptyset \succ_a \mu(a)$ *(or* $\emptyset \succ_c \mu(c)$*) should not exist.*

Definition 3 (Blocking pair)**.** *A matching* μ *is blocked by an applicant-college pair* (a, c) *if they prefer each other to the match they receive at* μ*. That is,* $c \succ_a \mu(a)$ *and* $a \succ_c a', \exists\, a' \in \mu(c)$.

Definition 4 (Stablility)**.** *A matching* μ *is* stable *if and only if it is both* individually rational *and not* blocked *by any other pairing between applicants and colleges.*

In other words, a matching of applicants to colleges will be *unstable* if there are two applicants a and a' who are matched to colleges c and c' respectively, although a' prefers c to c' and c prefers a' to a.

Gale and Shapley [3] has proved that a stable matching always exists, and that among possibly many stable matchings, the *optimal* matching—from the perspective of *applicants*—is the most desirable: a stable matching is *optimal* if every applicant is at least as well off under it as under any other stable matching. Gale and Shapley [3] also proved that the deferred acceptance algorithm that they proposed, with the applicants proposing, is optimal from the perspective of the applicants. The deferred acceptance algorithm is simple, runs with polynomial-time complexity, and can potentially be implemented in a decentralized fashion, as we shall show later in this paper.

A potential problem of using the deferred acceptance algorithm is that it can only produce two extreme outcomes, one that is applicant-optimal and one college-optimal [7]. This is known as polarization of stable matchings [12]. However, as we shall discuss, such polarization of stable matchings is not a significant problem in our context since applications in the metaverse are either latency-sensitive or bandwidth-intensive, but likely not emphasizing both.

3.2 Rendezvous through the lens of Stable Matching

We are now ready to answer the core question in this paper: how do we design a decentralized rendezvous service that produces a stable matching between the metaverse users and their servers? Let us formulate the problem first through the lens of stable matching.

Similar to the college admissions problem, we now have a set of n users, $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$, and a set of m geographically distributed servers, $\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$, each of which has its own ranking (preference) over its counterparts on the other side. We again use the notation $\succ_i$ to denote the ordering relationship in i's ranking (on either side of the metaverse network). If user i prefers to remain unmatched than being matched to college j, i.e. $\emptyset \succ_i j$, then j is said to be unacceptable to i. Again, there are no ties in the rankings.

How do we determine rankings from the perspectives of both metaverse users and their servers? From the perspective of an individual user, for latency-sensitive metaverse applications, we argue that the network latencies he/she experiences to each geographically distributed server are the most important factor, and as such network latencies should be used to produce his/her ranking of the servers. For bandwidth-intensive applications, however, we believe that the pair-wise available bandwidth between the user and each server is more important, and should be used to produce the ranking instead. From the perspective of a server, on the other hand, it is not immediately clear how it should rank the users, since it does not really have a preference for one user over another. As a tie breaker, we use the network latencies between the server and each user to produce the server's ranking, and we will later evaluate other alternatives to producing such a ranking in our experimental case study.

Despite their similarities, our problem at hand is quite different from the vanilla college admissions problem. In the original college admissions problem that is solved with the deferred acceptance algorithm, each college has a limited quota q_i—in terms of the number of open slots—for the applicants. In contrast, each server in our rendezvous service has a limited amount of resources, represented by its *bandwidth capacity*, which is the maximum bandwidth it can sustain when serving the users, and its *processing capacity*, which is the maximum number of virtual communities it can simultaneously process. The processing capacity is limited by the amount of hardware resources—such as CPU, memory, and storage resources—that the server has. Both types of server capacities may be *heterogeneous* across the servers. Each server can accommodate multiple users as long as these capacity constraints are observed. Such capacity constraints make it necessary to revise the vanilla definition of *blocking pairs* in Definition 3:

Definition 5 (Blocking pair with capacity constraints)**.** *A matching μ is blocked by a user-server pair (u, s) if they prefer each other to the match they receive at μ. That is,* both *of the following conditions hold:*

$$s \succ_u \mu(u), \tag{1}$$

$$\begin{aligned} & u \succ_s u', \exists\, u' \in \mu(s) \\ & \text{s.t. } B_u(s) - b(s,u) + b(s,u') \le B(s) \\ & \text{and } P_u(s) + \Delta p_s(u,u') \le P(s) \end{aligned} \tag{2}$$

where $B(s)$ and $P(s)$ denote the bandwidth and processing capacities of server s, respectively, $B_u(s)$ and $P_u(s)$ denote the bandwidth and processing capacities already utilized in the matching for s, $b(s,u)$ denotes the pair-wise available bottleneck bandwidth between s and u, and $\Delta p_s(u,u')$ denotes the additional processing needed for server s to switch from u to u'.

The formality in Definition 5 can be better understood with some explanations. There are two types of capacities at each server, and both need to be checked when we examine whether a pair u, s is a blocking pair that prevents a stable matching. With respect to bandwidth capacities, the server s being considered needs to compute the updated bandwidth demand by taking its currently utilized bandwidth $B_u(s)$ for its currently matched users, which can be easily measured or estimated in practice, subtracting the pair-wise bandwidth between s and one of its currently matched users u', and then adding the pair-wise bandwidth between s and u. The pair is only considered blocking when the updated bandwidth demand does not exceed its bandwidth capacity.

Similarly, with respect to processing capacities, s needs to consider the currently utilized processing capacity in terms of the number of communities it is currently serving, and calculate whether its processing capacity $P(s)$ will be exceeded if it removes u' and adds u to its matched users. Naturally, if u and u' belongs to the same virtual community, the currently utilized processing capacity does not change; this capacity constraint can only be violated when u and u' belong to different communities *and* s has already fully utilized its maximum processing capacity.

In this context, similar to the college admissions problem, a matching is considered *stable* between the metaverse users and their servers if there exist no blocking pairs subject to capacity constraints on the servers. As the nature of stable matching in our specific context is different from traditional college admissions, it is henceforth referred to as *stable rendezvous* in this paper.

3.3 Stable Rendezvous: Algorithm Design

Now that we have the precise definition of the stable rendezvous problem, we are ready to develop a revised version of the deferred acceptance algorithm to solve this problem. Before running this algorithm, we assume that each server's bandwidth capacity and its pair-wise available bandwidth to any user are both readily available. With respect to processing capacities, we make the simplifying assumption that they are linearly proportional to the number of communities that each server is ready to accommodate, so that when determining whether a user-server pair is blocking, we can just observe the number of communities a server accommodates before and after the switch of a user to its alternative. As input to the algorithm, rankings (preferences) on both the metaverse users and their servers are available; the former is determined based on network latencies, and the latter on pair-wise available bandwidth.

Algorithm 1 REVISED DEFERRED ACCEPTANCE ALGORITHM FOR STABLE RENDEZVOUS

```
Require: Set of servers and users $\mathcal{S}$ and $\mathcal{U}$, s.t. $\forall s \in \mathcal{S}, \forall u \in \mathcal{U}$; Servers' and users' rankings (preferences) $L(s)$, $L(u)$; Bandwidth capacity $B(s)$; Processing capacity $P(s)$; Pair-wise bandwidth between $s$ and $u$ $b(s,u)$; User $u$'s community ID $c(u)$
1:  while $\mathcal{N} \neq \emptyset$ do    ▷ $\mathcal{N}$ denotes the set of unassigned users
2:      $i \leftarrow 0$
3:      for $u \in \mathcal{N}$ do
4:          $s \leftarrow$ the $i^{\text{th}}$ item from $L(u)$
5:          Add $u$ to the set of users on hold at server $s$, $\mathcal{H}(s)$
6:      end for
7:      $\mathcal{N} \leftarrow \emptyset$
8:      for $s \in \mathcal{S}$ do
9:          for $u \in \mathcal{H}(s)$ do ▷ iterate in the ranked order in $L(s)$
10:             Add $c(u)$ into the set of community IDs $\mathcal{C}(s)$
11:             if $b(s,u) \le B(s)$ and $|\mathcal{C}(s)| \le P(s)$ then
12:                 $B(s) \leftarrow B(s) - B(s,u)$
13:             else
14:                 Remove $u$ from $\mathcal{H}(s)$
15:                 Add $u$ into the set of rejected users $\mathcal{R}(s)$
16:                 Remove $c(u)$ from $\mathcal{C}(s)$
17:             end if
18:         end for
19:         $\mathcal{N} = \mathcal{N} \cup \mathcal{R}(s)$
20:     end for
21:     $i \leftarrow i + 1$
22: end while
23: return the final matching $\mathcal{H}(s)$
```

Naturally, the design of our proposed algorithm is an extension of deferred acceptance for the college admissions problem. The main differences are proposed to accommodate both bandwidth and processing capacity constraints on the servers, since we no longer have the convenience of fixed-sized open slots for colleges to accommodate applicants in the original college admissions problem.

Our proposed algorithm, shown in Algorithm 1, proceeds as follows. As a starting point, all users contact the servers of their first choice using their respective preferences, $L(u)$. Each server s first places all the users who contacted it to its *hold set* $\mathcal{H}(s)$. Subject to its bandwidth and processing capacity constraints ($B(s)$ and $P(s)$), the server then places users who rank highest on its own preference list $L(s)$ on a waiting list. In order to determine who will be held on the waiting list, the server simply walks down its preferences, from the highest to the lowest ranked users, adding each user if it does not lead to a violation of either of the server's capacity constraints. Once the waiting list is established, the remaining users will be rejected in the first round.

In the second round, rejected users then contact their second choice. After a server has been contacted, it will start with a hold set that contains all the users on its waiting list in the previous round, and then add all the users who contacted it in the current round. Using this revised hold set, it will follow the same procedure as the previous round to determine its waiting list. This iterative algorithm terminates when every user is either in the waiting list of a server, or has been rejected by every server to which he/she is willing to contact (*i.e.*, in his/her ranking). At this point, each server admits everyone on its waiting list, and stable rendezvous has been achieved.

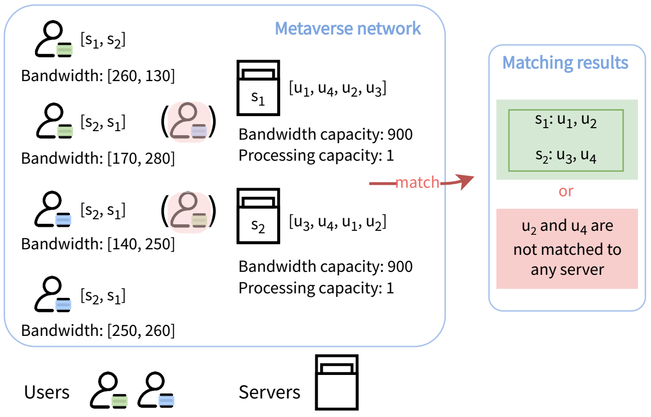


Fig. 2. As a toy example, we consider a metaverse network with four users and two servers. Each user maintains a ranking based on his/her pair-wise bandwidth to the servers. We use colours to indicate the community a user belongs to. The servers have their own bandwidth and processing capacities, and maintain a ranking of the users based on network latencies (the values of these latencies are not shown in the figure). We present two possible matching outcomes: the first is a normal outcome, and the second shows a pathological case that our deferred acceptance algorithm can fail to accommodate all the users even with sufficient server capacities.

3.4 A Toy Example

After our proposed deferred acceptance algorithm terminates, it is important to note that the bandwidth and processing capacities on some servers may not be fully utilized. A server's processing capacity is only related to the number of communities. Even a server with a low processing capacity can still support a large number of users in the same community, thereby shifting the server capacity bottleneck to its bandwidth capacity, which should not be exceeded by the sum of pair-wise available bandwidth from the matched users. We now show a toy example to show that, even with a sufficient amount of server capacity in both dimensions, we may still fail to achieve a stable matching that accommodates all the users.

Fig. 2 illustrates how metaverse users are matched to different servers based on their community interests and network conditions. In this example, the users produce their rankings based on pair-wise bandwidth to the servers, and have the community they belong to. Each user can only join one community at a time in the metaverse, and it is denoted by the blue or green colour in Fig. 2. Servers rank users by their network latencies, and have their respective bandwidth and processing capacity constraints. For instance, Server s_1 has a bandwidth capacity of 900 Mbps and a processing capacity of 1, with respect to the maximum number of communities it is able to serve.

Let us now run our deferred acceptance algorithm iteratively. In the first round, all users contact their highest ranked servers based on their preferences, and each server being contacted adds them to its hold set. s_1 has u_1 in its hold set, and s_2 has $[u_2, u_3, u_4]$. The servers should now check their respective bandwidth and processing capacity constraints, and produce their waiting lists: $[u_1]$ and $[u_3, u_4]$ for s_1 and s_2, respectively. At the end of the first round, the only user who is not on any waiting list is u_2. In the second round, u_2 contacts the next server on its preference, which is s_1. After being contacted, s_1 finds that it does have sufficient capacity to accommodate u_2, and adds it to its waiting list. When the algorithm terminates, all users have been matched to the servers: $\{(u_1, s_1), (u_2, s_1), (u_3, s_2), (u_4, s_2)\}$.

However, it should be noted that our deferred acceptance algorithm may lead to a remarkably different outcome if server capacity constraints and user communities change. In a pathological case, for example, if we change u_2's community to blue and u_3's community to green, the outcome of our algorithm will not accommodate all the users even with sufficient server capacities. In the first round, s_2 will hold only u_3 on its waiting list, and cannot hold u_4. Because the processing capacity of s_2 is 1, it only can accept users from the same community. Upon the termination of the algorithm, u_2 and u_4 are not matched to any server due to processing capacity constraints. Such a suboptimal outcome in this pathological case shows that the stable matching outcome produced by our deferred acceptance algorithm always prioritizes the rankings on both the users and the servers, and may not always fully utilize bandwidth and processing capacities on the servers. The stable matching may be optimal from the perspective of users, but not optimal when we examine the utilization of server capacities. In order to achieve the latter goal, we may need to consider formulating the problem as an optimization problem, which again is inherently centralized in nature.

3.5 Stablility

We now proceed to show that the outcome that our deferred acceptance algorithm produces is indeed a stable matching.

Theorem 1 (Stability)**.** *The user-server matching that Algorithm 1 produces is stable.*

Proof. Suppose a user u and a server s' are not matched to each other, by u prefers s' to its own matched server s. If this is the case, u must have contacted s' in some round of our iterative algorithm, and subsequently rejected by s' due to one of two potential reasons.

First, using its own ranking, the server s' is in favour of some other user u' that s' preferred to u, in which case s' must prefer u' to u, and the pair (u, s') is therefore not a blocking pair.

Second, the server s' rejects u since u's acceptance to its waiting list would violate either (or both) of the capacity constraints on s'. In our algorithm, s' may walk down its own preference list and add another user u', who is lower ranked than u, into its waiting list.

We argue, however, that based on our definition of the blocking pair in Definition 5, the pair (u, s') is still not a blocking pair as Eq. (2) is violated. This is because by switching u' back to u on server s', we are practically subtracting u''s usage of the bandwidth and processing capacity, moving the state of using these capacities on the server s' back to the step when s' decided to reject u. Since the only reason that s' rejected u was because the either or both of the capacity constraints on s' were violated, Eq. (2) must not hold.

$\square$

3.6 Stable Rendezvous: Decentralized Implementation

For the best possible scalability, it is important to have a fully decentralized implementation of our deferred acceptance algorithm. We argue that one of the salient advantages of our algorithm is that it is decentralized in nature, with the aid of message passing between the users and the servers. We now present a new implementation that is both *decentralized* and *asynchronous*, and that produces precisely the same stable rendezvous as Algorithm 1, which is inspired by existing work in the literature (*e.g.*, [7]).

In our decentralized implementation, each metaverse user begins by sending a `propose` message to propose to the server of its first choice, and then waiting for a reply. If his/her proposal has been accepted to the server's waiting list, the user will not hear anything back from the server, and will wait for further replies asynchronously. Whenever he/she receives a negative reply of `reject`, he/she will propose to the next server on his/her preference. If the preference list is exhausted, the user is considered unmatched and not included in the outcome of the algorithm. Otherwise, the user will eventually receive an affirmative reply of `accept`, implying that the servers have finalized their waiting lists, and the user will be considered matched to the last server he/she proposed to.

From the viewpoint of the servers, the implementation is slightly more complicated. Upon receiving a proposal from a user, each server will add it to its hold set $\mathcal{H}(s)$ first. After a predetermined timeout expires, it makes the decision to accept users from $\mathcal{H}(s)$ into its waiting list (by keeping them in $\mathcal{H}(s)$) or to reject it, using Algorithm 1. All users rejected will be sent a `reject` message, and the server continuously waits for more proposals in the next round. The server terminates its algorithm after m rounds, where m is the total number of servers in the metaverse, and sends an `accept` message to all the users who remain in its waiting list.

Our decentralized implementation of Algorithm 1 is fully asynchronous, as actions taken at both the metaverse users and their servers are completely driven by two kinds of events: the arrival of a message or the expiration of a timer. Each round on the server is guaranteed to finish within a pre-determined timeout period, and with a finite number of m rounds, termination of the server algorithm is guaranteed. New users can participate in the algorithm at any time by sending a message to its server of first choice, restarting the algorithm again. As our decentralized implementation shares the same deferred acceptance procedure, the final matching achieved when no one can make new proposals is exactly the same as its centralized counterpart.

3.7 Practical Implications

We conclude this section with some further remarks and discussions on how our proposed algorithm can be implemented in practice, when real-world metaverse users and servers are involved.

The Use of timers. Due to the use of timers in our decentralized implementation, the servers are at risk of starting to reject users and to progress to the next round without waiting for users who are slower to initiate the `propose` message in the current round. We argue that this is not an issue in practice, as the server can always consider this proposal in its next round, and reject users who are on its waiting list at that time. These rejected users will propose to their next-best choices in their own rankings in later rounds.

Processing capacities on the servers. Algorithm 1 makes a simplifying assumption that the processing capacity on the servers can be evaluated by counting the number of communities that the server is supporting. One can argue that as the number of users in the same community increases, server resource consumption may increase proportionally, thereby "consuming" the server's processing capacity. This is a valid observation in practice, but

Algorithm 1 can easily be extended to accommodate this case in practice, by imposing a bound on the maximum number of users in each community supported by the server. In general, our algorithm only assumes that the server is able to estimate both its capacity constraints—in terms of both bandwidth and processing capacities—and to predict the "consumption" of such capacities from each user when he/she proposes. This assumption is quite mild and typically holds in practice in the metaverse.

Pair-wise bandwidth. Algorithm 1 also makes the assumption that pair-wise available bandwidth between a user and a server is readily available as input. In practice, crude estimates of pair-wise available bandwidth can be achieved using any of the bandwidth estimation algorithms, such as pathChirp [13]. We believe that bandwidth estimates do not need to be accurate for the algorithm to be effective: even just a binary estimate—"sufficient" and "low" for example—may be good enough for a user to avoid servers that are not suitable to meet its bandwidth needs. More accurate estimates are obviously desirable, especially for bandwidth-intensive applications in the metaverse.

4 Multi-Server Federated Learning: An Experimental Case Study

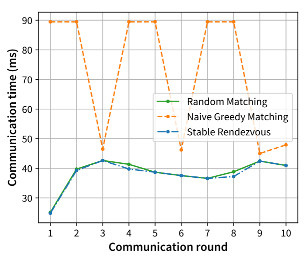
The privacy concerns of metaverse users are paramount [5]. As such, it is necessary to ensure that personal activities and financial transactions within the metaverse are kept strictly with the user. To evaluate the feasibility and performance of the rendezvous service and our proposed algorithm, we present an experimental case study using *multi-server federated learning* [14].

As a distributed machine learning paradigm that allows a large number of users to collaboratively train a shared machine learning (ML) model while preserving the privacy of user data, federated learning (FL) [4] has the potential to improve the user experience in the metaverse by training ML models to be used in metaverse applications. For instance, FL can be used to train an ML model to predict network latencies or to optimize the rendering of virtual objects. Several existing works in the literature have studied the use of federated learning in the metaverse (*e.g.*, [15]).

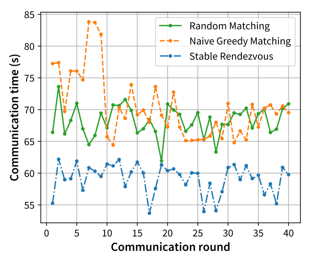
Our experimental case study involves training tasks using the paradigm of *multi-server federated learning* [14]. Departing from the conventional wisdom of using a single FL server, multi-server FL allows multiple FL servers to be deployed concurrently, each serving a subset of users. Our experimental case study involves a careful implementation of Algorithm 1 in the context of multi-server FL, which we developed using Plato a scalable open-source FL framework that is designed to evaluate new FL algorithms involving a large number of users, yet within the confines of one multi-GPU server.

Implementing the deferred acceptance algorithm. Our implementation of Algorithm 1 utilizes randomly generated a pair-wise bandwidth matrix and delay matrix between the users and the servers, and then simulating multiple servers using one actual server process. The users communicate with the server process using the real-world WebSockets protocol, which is supported by `Plato`. We have also implemented a procedure to verify the stability of the outcome produced by our algorithm, using the definition of blocking pairs in Definition 5. Our implementation will be made available as open-source.

In a general FL workflow, each server selects a subset of clients and sends them a copy of the global model. The user trains the model locally using its own data, and sends updated model



(a) Multi-server FL with `MNIST`.



(b) Multi-server FL with `CIFAR-10`.

Fig. 3. Communication time taken in each of the communication rounds.

parameters to its server. This workflow proceeds in asynchronous communication rounds with a buffer of updates from clients, until the model converges to a target accuracy. As the size of ML models is typically large, such a multi-server FL workflow is a typical bandwidth-intensive application. Our algorithm is executed periodically, due to the dynamic nature of metaverse users, who may be participating in different communities and experiencing varying available bandwidth and latencies over time.

Experimental settings. To evaluate the performance of our rendezvous service, we compared it with two baseline heuristics. The first baseline is a simple random matching algorithm, which assigns each user to a randomly selected server. The second baseline alternative is a naïve greedy algorithm, which assigns each user to the server that offers the highest pair-wise bandwidth between them. The community that each user participates in is generated randomly with a uniform distribution, and both the pair-wise available bandwidth and network latencies are both drawn from normal distributions: with 100 KB/second and 300 milliseconds as the mean, and 10 KB/second and 100 milliseconds as the standard deviation, respectively. We deployed 5 servers, with a total of 500 users and 15 users selected per round, and our evaluations are carried out (1) using the `MNIST` dataset to train the `LeNet-5` model; and (2) using the `CIFAR-10` dataset to train the `ResNet-18` model.

Experimental results.

Evaluating the training time in communication rounds. The time it takes to complete each communication round in multi-server FL training depends on both the computation time training the model locally on each user, and the time it takes to transmit the model to and from the server. The latter naturally depends heavily on the amount of bandwidth available. With random and naïve greedy matching algorithms, a server can be over-subscribed with more users, violating its capacity constraints. In such cases, the server will have no choice but to share its limited bandwidth capacity across all the users it is currently serving, resulting in a lower achievable throughput for each user.

Figs. 3a and 3b show the communication time taken in each of the communication rounds, comparing our proposed algorithm with both random and naïve greedy heuristics. We observe that the naïve greedy algorithm experienced more diverse and longer round times over different communication rounds. This is because when users are always matched to the server with the highest pair-wise bandwidth, over-subscription of server capacities tends to be more severe in some of the rounds than simply using a random matching algorithm. In contrast, the stable rendezvous outcomes computed by our proposed algorithm experienced per-round times that were the most steady over time. Most notably, our stable rendezvous algorithm significantly reduces communication time,

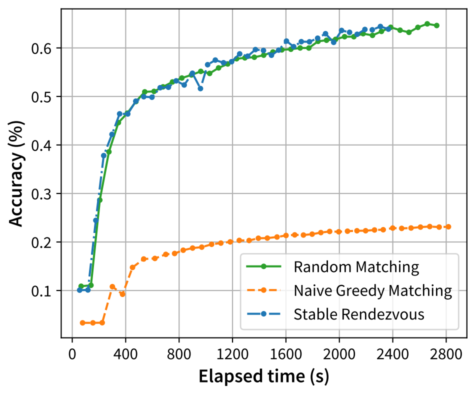


Fig. 4. Validation accuracy of the global model over the elapsed wall-clock time in multi-server FL with `CIFAR-10`.

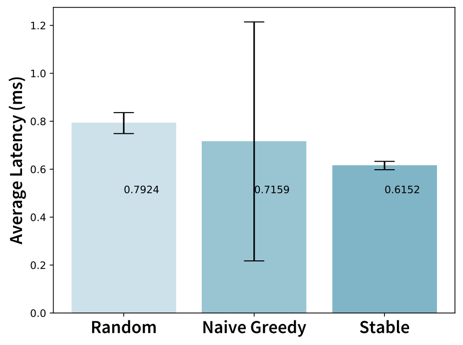


Fig. 5. Average network latencies across the users, produced by stable rendezvous, random, and naïve greedy algorithms.

particularly on the `CIFAR-10` dataset. This is because the payload of the `MNIST` task is relatively small and the bandwidth influence is less pronounced than for larger tasks such as `CIFAR-10`.

Evaluating the behaviour for the global accuracy to converge over time. In addition to the communication time in each round, we also wish to show how the global model converges over time, with respect to the validation accuracy of the global model. Fig. 4 show the average validation accuracy across multiple servers, as models on these servers converge over time, in a training session using asynchronous federated learning. While our algorithm with stable rendezvous shows a decisive advantage over naïve greedy matching, the advantage over random matching is not as substantial but still easily noticeable, as stable rendezvous converges over 300 seconds faster than random matching. This indicates that our algorithm produces a better allocation of server bandwidth to users, and such advantages have indeed translated to a better overall performance in multi-server asynchronous FL training.

Evaluating network latencies. Naturally, the average latency experienced by all users may also be affected by how users are matched to the servers. The lower latencies with stable rendezvous, compared to its competitors, can be attributed to the fact that stable rendezvous takes latency into account on the server side, using it as a basis for matching users (as indicated in the server's ranking of users). This demonstrates the effectiveness of incorporating latencies into the matching process to reduce the overall latency for users.

Finally, it is important to note that while stable rendezvous may not offer a substantial performance advantage over its random or greedy counterparts in our experiments, it is also exceedingly simple and fully decentralized, adding no unnecessary implementation complexity as compared to a random or greedy algorithm. In addition, we believe that the performance benefits offered by stable rendezvous will become more substantial as the number of users scale up to levels well beyond what we are able to accommodate in our experimental settings due to limitations of GPU resources.

5 Concluding Remarks

In this paper, we advocated for a new scalable rendezvous service between millions of users in the metaverse and a collection of geographically distributed servers, in order to best accommodate the low-latency and high-bandwidth requirements in metaverse applications. We proposed to use stable matching theory to design a revised deferred acceptance algorithm to achieve stable rendezvous, taking server capacity constraints into account. In contrast to conventional optimization-based and game-theoretic solutions, our algorithm is simple, fully decentralized and asynchronous, enjoys polynomial-time complexity, and produces an outcome that is stable and optimal from the perspective of metaverse users. We have experimentally evaluated the feasibility and performance of our algorithm in the case study of multi-server federated learning, and show that it outperformed random and greedy heuristics without inducing additional complexity.

References

[1] J. D. N. Dionisio, W. G. B. III, and R. Gilbert, "3D Virtual Worlds and the Metaverse: Current Status and Future Possibilities," *ACM Computing Surveys (CSUR)*, vol. 45, no. 3, pp. 1–38, 2013.

[2] H. Duan, J. Li, S. Fan, Z. Lin, X. Wu, and W. Cai, "Metaverse for Social Good: A University Campus Prototype," in *Proc. International Conference on Multimedia*. ACM, 2021, pp. 153–161.

[3] D. Gale and L. S. Shapley, "College Admissions and the Stability of Marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[4] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proc. Artificial Intelligence and Statistics (AISTATS)*, 2017.

[5] Y. Wang, Z. Su, N. Zhang, R. Xing, D. Liu, T. H. Luan, and X. Shen, "A Survey on Metaverse: Fundamentals, Security, and Privacy," *IEEE Communications Surveys and Tutorials*, 2022.

[6] S. Chieochan and E. Hossain, "Channel Assignment for Throughput Optimization in Multichannel Multiradio Wireless Mesh Networks Using Network Coding," *IEEE Transactions on Mobile Computing*, vol. 12, no. 1, pp. 118–135, 2013.

[7] H. Xu and B. Li, "Seen as Stable Marriages," in *Proc. IEEE INFOCOM*. IEEE, 2011, pp. 586–590.

[8] A. A. Roth and M. A. O. Sotomayor, *Two-Sided Matching: A Study in Game Theoretic Modeling and Analysis*. Cambridge University Press, 1990, vol. 18.

[9] H. Xu and B. Li, "Anchor: A Versatile and Efficient Framework for Resource Management in the Cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1066–1076, 2012.

[10] Y. Feng, B. Li, and B. Li, "Bargaining Towards Maximized Resource Utilization in Video Streaming Datacenters," in *Proc. IEEE INFOCOM*. IEEE, 2012.

[11] W. Wang, B. Li, and B. Liang, "Bargaining Towards Maximized Resource Utilization in Video Streaming Datacenters," in *Proc. 8th IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*. IEEE, 2011.

[12] A. E. Roth, "The College Admissions Problem Is Not Equivalent to the Marriage Problem," *Journal of economic Theory*, vol. 36, no. 2, pp. 277–288, 1985.

[13] V. Ribeiro, R. Riedi, J. Navrátil, and L. Cottrell, "pathChirp: Efficient Available Bandwidth Estimation for Network Paths," in *Proc. Passive and Active Measurement Workshop*, April 2003.

[14] T. Taleb, I. Afolabi, K. Samdanis, and F. Z. Yousaf, "On Multi-Domain Network Slicing Orchestration Architecture and Federated Resource Control," *IEEE Network*, vol. 33, no. 5, pp. 242–252, 2019.

[15] X. Zhou, C. Liu, and J. Zhao, "Resource Allocation of Federated Learning for the Metaverse with Mobile Augmented Reality," *arXiv preprint arXiv:2211.08705*, 2022.