Cost-Effective Low-Delay Cloud Video Conferencing

Mohammad H. Hajiesmaili*‡, Lok To Mak†, Zhi Wang§, Chuan Wu¶, Minghua Chen†, and Ahmad Khonsari*‖
* School of ECE, College of Engineering, University of Tehran, Iran
† Department of Information Engineering, The Chinese University of Hong Kong
‡ Institute of Network Coding, The Chinese University of Hong Kong
§ Graduate School of Shenzhen, Tsinghua University
¶ Department of Computer Science, The University of Hong Kong
‖ School of Computer Science, IPM, Iran
Emails: {mohammad,minghua,mlt014}@ie.cuhk.edu.hk,wangzhi@sz.tsinghua.edu.cn,cwu@cs.hku.hk,ak@ipm.ir

***Abstract*—The cloud computing paradigm has been advocated in recent video conferencing system design, which exploits the rich on-demand resources spanning multiple geographic regions of a distributed cloud, for better conferencing experience. A typical architectural design in cloud environment is to create video conferencing *agents*, i.e., virtual machines, in each cloud site, assign users to the agents, and enable inter-user communication through the agents. Given the diversity of devices and network connectivities of the users, the agents may also transcode the conferencing streams to the best formats and bitrates. In this architecture, two key issues exist on how to effectively assign users to agents and how to identify the best agent to perform a transcoding task, which are nontrivial due to the following: (1) the existing proximity-based assignment may not be optimal in terms of inter-user delay, which fails to consider the whereabouts of the other users in a conferencing session; (2) the agents may have heterogeneous bandwidth and processing availability, such that the best transcoding agents should be carefully identified, for cost minimization while best serving all the users requiring the transcoded streams.**

To address these challenges, we formulate the user-to-agent assignment and transcoding-agent selection problems as an optimization problem, which targets at minimizing the operational cost of the conferencing provider while keeping the conferencing delay low. The problem is combinatorial in nature and difficult to solve in static settings and more challenging in dynamic settings. Using the recently-proposed Markov approximation framework of tackling combinatorial problems, we design a decentralized algorithm that provably converges to a bounded neighborhood of the optimal solution and adapts to system dynamics. An agent ranking scheme is also proposed to properly initialize our algorithm so as to improve its convergence. We implement a prototype video conferencing system realizing our algorithms, and carry out extensive evaluations using real-world traces. In a set of Internet-scale scenarios, our design reduces the operational cost by 77% as compared to a commonly-adopted alternative, while simultaneously yielding lower conferencing delays.

I. Introduction

As front-facing cameras become popular on personal devices (*e.g.*, laptops, tablets, and smart phones), recent years have witnessed a skyrocketing growth of video conferencing (VC) systems on those devices. According to Cisco, the number of video conferencing users is growing at an annual rate of 51.7% and will surpass that of audio conferencing users by 2015 [2]. Another trend has been the advocation of cloud computing services in multi-party VC systems, to overcome the constraints of user devices and boosting the conferencing experience, employing the rich and on-demand resources provided by a geo-distributed cloud platform.

(a) Clinet/Server (b) Peer-to-Peer (c) Cloud

Fig. 1. Different VC architectures

In a typical cloud-assisted VC system design [10], [17], illustrated in Fig. 1(c), video conferencing *agents*, i.e., virtual machines, are created in each cloud site, and *users* join a conferencing *session* by subscribing to those cloud agents. Users communicate through the agents, which exchange conferencing streams among each other, transcode the streams to the best formats and bitrates and deliver them to users with diverse devices and network connectivities. Such a cloud-assisted VC paradigm outperforms traditional client/server (C/S) based (Fig. 1(a)) and P2P-based (Fig. 1(b)) VC approaches, due to the following:

(i) Meeting stringent delay requirements better. According to ITU-T Recommendation G.114 [11], the maximum acceptable user-to-user conferencing delay is 400 *ms*. In a C/S architecture, clients may often suffer from a long delay due to considerable distances from the servers, due to limited deployment of the costly infrastructure. Direct connections between users in a P2P system may yield lower delays, while measurements [10] have corroborated that the delay in a cloud-assisted VC system is comparable or even lower than that.

(ii) Providing more bandwidth and computation capacity at lower costs. Conferencing devices are diverse in screen resolution ($\approx$ 100 possible resolutions), hardware ($\approx$ 2800 types), and OS ($\approx$ 14 types) [14]. On-the-fly *transcoding* is demanded for converting the streams from one format/bitrate to another, to cater for such device heterogeneity. The C/S architecture utilizes dedicated servers, but suffers from limited scalability and high operational costs. The limited capacity of peers in the P2P design hinders such computation-intensive jobs, and hence the number of peers allowed in a VC session is often significantly limited. In contrast, cloud-assisted VC provides scalability by employing on-demand bandwidth/computation resources at cloud agents, at a lower cost.

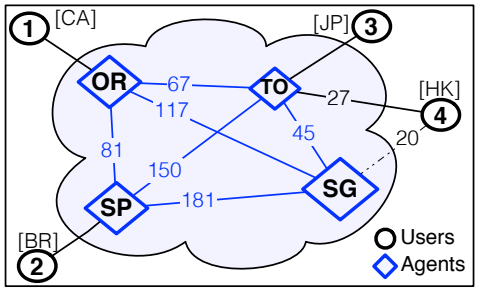


Fig. 2. A VC scenario with 4 users (PlanetLab nodes) in 1 session and 4 cloud agents (EC2 instances). Values on edges are real-world measured latencies. Agents in larger diamonds have higher capabilities. SG: Singapore, TO: Tokyo, OR: Oregon, SP: Sao Paulo.

Nevertheless, two key challenges still exist in the state-of-the-art design of cloud-assisted VC, for optimizing both the operational cost of the service provider and the conferencing experience of the users. *First*, current design typically assigns users to the nearest agents in terms of delay [10], [17], which may not be optimal in inter-user delay and traffic cost, as they are oblivious to whereabouts of the other users in a conferencing session and diversity of transcoding latency in heterogeneous agents. For example in Fig. 2, user 4 should be assigned to SG agent following the nearest assignment policy. However, assigning user 4 to TO agent is better since: (i) the end-to-end flow delays in this session are reduced because TO agent is closer to the other 3 agents than SG agent, e.g., the delay of flow from user 4 to user 1 via TO is at least $27+67$ while the delay via SG is at least $20+117$; (ii) since user 3 is assigned to TO agent, assigning user 4 to TO eliminates any inter-agent stream exchanges with SG agent, leading to reduced traffic cost as well.

Second, how to identify the best agent to perform a transcoding task, given the heterogeneity of agent VMs, has not been well studied in the literature. The agents may have diverse resource availability, leading to different transcoding delays. The best transcoding agents should be carefully identified, for cost minimization while best serving all the users requiring the transcoded streams. For instance in Fig. 2, though we have shown assigning user 4 to TO agent leads to lower delay and traffic cost, SG agent is better in terms of transcoding delay, given that it is more computationally powerful than TO agent.

All the existing studies we are aware of and review in Sec. V adopt the nearest policy for user-to-agent assignment [10], [17]. To the authors' knowledge, this work is the first to simultaneously minimize the service provider's cost and maximize the user's experience in a cloud-assisted conferencing system, by addressing user-to-agent assignment and transcoding task assignment problems in a unified mathematical framework. The main contributions of the paper are summarized as follows:

▷ We formulate the **User-to-agent Assignment Problem** (**UAP**) (Sec. II), which finds the best user-to-agent assignment and transcoding task assignment solution to minimize the overall cost of the service provider and inter-user delay at the same time, subject to capacity constraints of the heterogeneous agents and end-to-end delay constraints of the users. The problem is a nonlinear combinatorial optimization problem, difficult to solve even in the centralized manner under static system settings.

▷ Inspired by the Markov approximation approach [7] which is a nice technique to construct parallel and adaptive solutions, we devise an efficient distributed algorithm to solve **UAP**, which runs locally in each session and optimizes the overall assignment (Sec. III-A). Highlights of the algorithm are its adaptability to system dynamics, bounded approximation gap, and robustness in case of inaccurate measurements of transcoding latencies and RTT between nodes.

▷ We propose a *proximity- and resource-aware agent ranking* scheme, called *AgRank*, as the initialization step of our algorithm, which further improves the convergence of the algorithm (Sec. III-B). The scheme features a high success rate for the initial user-to-agent assignment, i.e., the initial assignment by *AgRank* significantly overlaps with the optimal assignment when the entire algorithm is completed.

▷ We implement a system prototype and carry out trace-driven evaluation experiments using PlanetLab nodes and Amazon EC2 instances (Sec. IV). Observations from the experiments demonstrate the significant improvement brought by our solution in both static and dynamic scenarios. In a set of typical Internet-scale scenarios, our solution simultaneously reduces the traffic cost and the delay by 77% and 2%, respectively, as compared to the commonly-adopted nearest assignment strategy.

II. Problem Formulation

Consider a cloud-assisted video conferencing system with multiple conferencing sessions, each of which is established among a set of users. Each user in a session records a video in a specific format/bitrate/resolution (referred to as a *representation*), streams it to other users via cloud agents, and demands streams of specific representations from the other participants. Along each *flow* from a source user to a destination user, the upstream representation produced by the source may be different from the downstream representation required by the destination, and *transcoding* is carried out at the agents. We proceed with detailed definitions of elements of our model using the key notation in Table I.

Session and user. Let $\mathcal{S}$ be the set of sessions and $\mathcal{U}$ be the set of users. Assuming that each user participates in exactly one session, we denote the users of session s by $\mathcal{U}(s) \subseteq \mathcal{U}$ and the session that user u belongs to by $s(u) \in \mathcal{S}$. Let $\mathcal{P}(u) \subseteq \mathcal{U}$ be the set of other participants in user u's session, (i.e., $\mathcal{P}(u) = \{v | v \in \mathcal{U}, s(v) = s(u), v \neq u\}$).

Representation. A representation refers to a specific configuration of format, encoding bitrate and spatial/temporal resolution of a stream, e.g., example representations of YouTube videos are (360p, 1 Mbps), (480p, 2.5 Mbps), (720p, 5 Mbps), (1080p, 8 Mbps), etc. Let $\mathcal{R}$ be the set of all possible representations of all the users. Based on the access bandwidth and hardware specification of the device, each user specifies its *upstream* representation, $r_u^{\mathbf{u}} \in \mathcal{R}$, which is the representation of the stream it produces, and *downstream* representation, $r_{uv}^{\mathbf{d}} \in \mathcal{R}$, which is its required representation of the stream from another user v in the session. Let $\kappa(r)$ denote the corresponding bit-rate of representation r. We also define $\theta = [\theta_{uv}]_{U \times U}$ as the *transcoding matrix*, where $\theta_{uv} = 1$

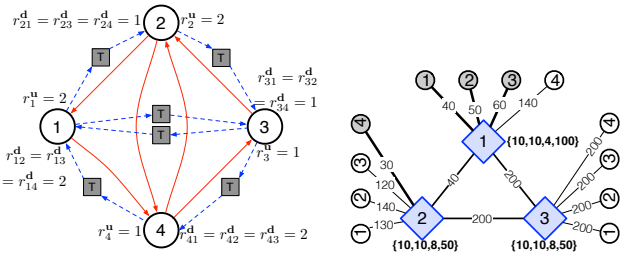
	Notation	Definition
Users	$\mathcal{S}$	Set of VC sessions, $S \triangleq \lvert\mathcal{S}\rvert$
	$\mathcal{U}$	Set of users, $U \triangleq \lvert\mathcal{U}\rvert$
	$\mathcal{U}(s)$	Users of session s
	$s(u)$	Session of user u
	$\mathcal{P}(u)$	Set of other participants in user u's session
Representation	$\mathcal{R}$	Set of video representations, $R \triangleq \lvert\mathcal{R}\rvert$
	$\kappa(r)$	Corresponding bit-rate of representation r
	$r_u^{\mathbf{u}}$	Upstream representation of user u
	$r_{uv}^{\mathbf{d}}$	Downstream repr. of user u from user v
	θ	$U \times U$ transcoding matrix
Agents	$\mathcal{L}$	Set of cloud agents, $L \triangleq \lvert\mathcal{L}\rvert$
	u_l	Upload capacity of agent l
	d_l	Download capacity of agent l
	t_l	Transcoding capacity of agent l
	σ_l	Transcoding latency of agent l
	D	$L \times L$ inter-agent delay matrix
	H	$L \times U$ agent-to-user delay matrix
Opt. Vars.	λ_{lu}	User assignment variable; 1 if user u is assigned to agent l, 0 otherwise
	γ_{lruv}	Transcoding task assignment variable; 1 if $r_{vu}^{\mathbf{d}} = r$ and the transcoding is done at agent l, 0 otherwise

TABLE I
KEY NOTATIONS

if source u and destination v are in the same session but produce/require different representations, i.e., $s(v) = s(u)$ and $r_u^{\mathbf{u}} \neq r_{vu}^{\mathbf{d}}$, and $\theta_{uv} = 0$, otherwise[1].

Cloud agent. Agents, in set $\mathcal{L}$, are virtual machines which the VC service provider leases from disparate cloud sites (data centers) in advance. Each agent $l \in \mathcal{L}$ is described by a quadruple $\{u_l, d_l, t_l, \sigma_l(.)\}$, corresponding to its upload bandwidth capacity (in Mbps), download bandwidth capacity (in Mbps), transcoding capacity (the number of concurrent transcoding tasks), and transcoding latency (in ms), respectively. We assume that each agent allocates a fixed amount of resources (CPU, memory) for each transcoding task, i.e., one unit of its transcoding capacity, such that its number of concurrent transcoding tasks can be derived. The transcoding latency $\sigma_l(r_1, r_2)$ is an increasing function of the bit-rates of the input (r_1) and output (r_2) representations, given that transcoding is typically done by decoding the source stream to an intermediate format, and then re-encoding the stream from the intermediate format to the destination bit-rate [17]. We assume that the VC provider obtains agent-to-user and inter-agent delays through active measurements. Let $D = [D_{lk}]_{L\times L}$ be the *inter-agent delay matrix* and $H = [H_{lu}]_{L\times U}$ be the *agent-to-user delay matrix*, where D_{lk} is the latency between agents l and k and H_{lu} is the propagation delay between agent l and user u. We have $D_{ll} = 0$ and $D_{lk} = D_{kl}$, $\forall l, k \in \mathcal{L}$, and we assume the triangle inequality holds for inter-agent delays, i.e., $D_{lk} < D_{ll'} + D_{l'k}$, $\forall l, l', k \in \mathcal{L}$, thereby inter-agent transmission is done by direct transmission between agents. In practice, these values are subject to change, and thus a proper design should be robust to tolerate noisy measurements (Sec. III-A3).

An illustrative example. Fig. 3 illustrates a conferencing session with 4 users and 3 agents. In Fig. 3(a), when the upstream representation differs from the downstream representation, a flow is marked using dotted lines with a transcoding task in the middle (e.g., the flow from user 1 to user 3); flows where the upstream and downstream representations are the same are depicted using solid lines (e.g., flows between user 2 and user 4). Fig. 3(b) plots a potential user-to-agent assignment by highlighting the assigned links with thick lines, i.e., users 1, 2 and 3 are assigned to agent 1 and user 4 is assigned to agent 2.



(a) Users are labeled by their representations. Squares denote transcoding operations.

(b) Agents in diamonds are labeled with quadruple $\{u_l, d_l, t_l, \sigma_l\}$. Links are labeled with delay.

Fig. 3. A VC Scenario: $S = 1, U = 4, L = 3, \mathcal{R} = \{1, 2\}$.

Optimization variables. Let λ_{lu} be the *user assignment* variable such that $\lambda_{lu} = 1$ if user u is assigned to agent l, and $\lambda_{lu} = 0$, otherwise. Each user must subscribe to exactly one agent. Hence λ_{lu}'s satisfy the following:

$$\sum_{l\in\mathcal{L}} \lambda_{lu} = 1, \qquad \forall u \in \mathcal{U}, \tag{1}$$

$$\lambda_{lu} \in \{0, 1\}, \qquad \forall l \in \mathcal{L}, \forall u \in \mathcal{U}. \tag{2}$$

Another category of decisions is which agents should perform which transcoding tasks. The transcoding from an upstream representation to a different downstream representation can potentially be done at the *source agent*, the *destination agent*, or a *tertiary agent*.[2] Let γ_{lruv} be the *transcoding task assignment* variable where $\gamma_{lruv} = 1$ if user v requires representation r from user u (i.e., $r_{vu}^{\mathbf{d}} = r$) and the transcoding is done at agent l, and $\gamma_{lruv} = 0$, otherwise. γ_{lruv}'s satisfy the following constraints:

$$\sum_{l\in\mathcal{L}} \sum_{r\in\mathcal{R}} \gamma_{lruv} = \theta_{uv}, \forall u \in \mathcal{U}, \forall v \in \mathcal{P}(u), \tag{3}$$

$$\gamma_{lruv} \in \{0, 1\}, \forall l \in \mathcal{L}, \forall r \in \mathcal{R}, \forall u \in \mathcal{U}, \forall v \in \mathcal{P}(u). \tag{4}$$

Constraint (3) states that transcoding of the flow from u to v is needed only when $\theta_{uv} = 1$, i.e., the upstream and downstream representations differ, and exactly one agent should carry out the transcoding to the required representation.

The dimension of our decision space is $O(L^{U+\theta^{\text{sum}}})$, where U, θ^{sum}, and L are the total numbers of the users, the transcoding tasks, and the agents, respectively.

Download and upload capacity constraints. For notational convenience, let $\nu_{lru} \triangleq \max_{v\in\mathcal{P}(u)} \gamma_{lruv}$ denote whether agent l transcodes u's stream to representation r for at least one other participant in u's session (1 yes and 0 no), and $\nu'_{lu} \triangleq \max_{r\in\mathcal{R}} \nu_{lru}$ denote whether agent k transcodes u's

[1]Note that θ could be customized to support just high to low quality transcoding operations by changing the definition of $\theta_{uv} = 1$ as $s(v) = s(u)$ and $r_{vu}^{\mathbf{down}} < r_u^{\mathbf{u}}$, by assuming ordered set of representations in quality.

[2]We do not consider possible parallel transcoding of the same flow at multiple agents in this work.

stream at all (1 yes and 0 no). The download capacity constraint of agent l is formulated as

$$\sum_{u\in\mathcal{U}}\Big(\lambda_{lu}\kappa(r_u^{\mathbf{u}}) + \sum_{k\in\mathcal{L},k\neq l}\mu_{klu}\Big) \leq d_l, \forall l \in \mathcal{L}, \quad (5)$$

where the first term is due to the last-mile upstream of users who directly subscribe to agent l and the second term depicts the outgoing traffic of user u from all other agents towards agent l. Define μ_{klu} to represent the download traffic at agent l due to receiving via another agent k the stream originated from user u, as follows:

$$\mu_{klu} = \lambda_{ku}\nu'_{lu}\kappa(r_u^{\mathbf{u}}) + (\max_{\substack{v\in\mathcal{P}(u),\\ \theta_{uv}=0}}\lambda_{lv})\lambda_{ku}(1-\nu'_{lu})\kappa(r_u^{\mathbf{u}}) + \sum_{\substack{r\in\mathcal{R},\\ r\neq r_u^{\mathbf{u}}}}(\max_{\substack{v\in\mathcal{P}(u),\\ r_{vu}^{\mathbf{d}}=r}}\lambda_{lv})(1-\lambda_{lu})\nu_{kru}\kappa(r),$$

where the first term represents the traffic from u's agent k to agent l for transferring u's stream for transcoding at l, the second term depicts the traffic of sending the upstream to other parties, and the last term is the traffic by considering bit-rate changes after transcoding. Similar to the download capacity constraint we get the following constraint for the upload capacity:

$$\sum_{u\in\mathcal{U}}\Big(\lambda_{lu}\sum_{v\in\mathcal{P}(u)}\kappa(r_{uv}^{\mathbf{d}}) + \sum_{k\in\mathcal{L},k\neq l}\mu_{lku}\Big) \leq u_l, \forall l \in \mathcal{L}, \quad (6)$$

Transcoding capacity constraints. Regardless of the number of destinations, transcoding of user u's upstream representation to representation r occupies one unit of the transcoding capacity of agent l. Hence the transcoding capacity constraint at l is formulated as follows:

$$\sum_{u\in\mathcal{U}}\sum_{r\in\mathcal{R}}\nu_{lru} \leq t_l, \quad \forall l \in \mathcal{L}. \quad (7)$$

End-to-end delay constraints. The end-to-end delay of a flow from user u to user v is the aggregation of the following: (1) propagation delay from u to u's agent l, H_{lu}; (2) the propagation delay between u's agent and v's agent, including two cases: (a) from u's agent l to v's agent k directly, D_{lk}, or (b) from u's agent l to a tertiary agent m (for transcoding) and then to v's agent k, $D_{lm}+D_{mk}$; (3) from v's agent k to v, H_{kv}; (4) (possibly) the transcoding latency at an agent l, $\sigma_l(r_u^{\mathbf{u}}, r_{vu}^{\mathbf{d}})$. We ignore any queuing delay at the agents, since our bandwidth and transcoding capacity constraints have ensured the availability of resources for the respective tasks. Employing the transcoding matrix θ and defining $\bar{\theta}_{uv} = 1-\theta_{uv}$, we get the end-to-end delay of flow $u \to v$ as

$$d_{uv} = \sum_{l\in\mathcal{L}}(\lambda_{lu}H_{lu} + \lambda_{lv}H_{lv}) + \bar{\theta}_{uv}\left(\sum_{l\in\mathcal{L}}\sum_{k\in\mathcal{L}}\lambda_{lu}\lambda_{kv}D_{lk}\right) + \theta_{uv}\left(\sum_{l\in\mathcal{L}}\sum_{k\in\mathcal{L}}\sum_{r\in\mathcal{R}}\gamma_{lruv}\Big(D_{lk}(\lambda_{ku}+\lambda_{kv}) + \sigma_l(r_u^{\mathbf{u}}, r_{vu}^{\mathbf{d}})\Big)\right).$$

Let $D^{\max}$ be the maximum acceptable delay, e.g., 400 *ms*. The end-to-end conferencing delay constraint is:

$$d_{uv} \leq D^{\max}, \quad \forall u \in \mathcal{U}, \forall v \in \mathcal{P}(u). \quad (8)$$

Objective function. We seek to minimize the overall operational cost of the VC service provider, as well as a delay cost based on inter-user delays. The operational cost of the provider contains two parts. (1) Inter-agent bandwidth costs: bandwidth cost of session s is formulated as $G(\boldsymbol{x}_s) = \sum_{l\in\mathcal{L}} g_l(x_{ls})$, where $x_{ls} = \sum_{u\in\mathcal{U}(s)}\sum_{k\in\mathcal{L},k\neq l}\mu_{klu}$ is the total incoming traffic to agent l from other agents in session s, and vector $\boldsymbol{x}_s = [x_{ls}]_{l\in\mathcal{L}}$. $g_l(.)$ is a convex and increasing function. Such a bandwidth cost only considers inter-agent data transfer, but not the last-mile traffic to/from users, since the latter is fixed in all possible user-to-agent assignments. (2) Transcoding cost at the agents: the overall transcoding cost in session s is similarly formulated as follows, where y_{ls} indicates the number of transcoding tasks agent l performs in this session and $h_l(.)$ is a convex function

$$H(\boldsymbol{y}_s) = \sum_{l\in\mathcal{L}} h_l(y_{ls}), \quad \boldsymbol{y}_s = [y_{ls}]_{l\in\mathcal{L}}, \quad y_{ls} = \sum_{u\in\mathcal{U}(s)}\sum_{r\in\mathcal{R}}\nu_{lru}.$$

The delay cost at users in session s is described by function $F(\boldsymbol{d}_s)$, where $\boldsymbol{d}_s = [d_u]_{u\in\mathcal{U}(s)}$, $d_u = \max_{v:u\in\mathcal{P}(v)} d_{vu}$ is the maximum end-to-end delay experienced by user u for receiving streams from other participants, and $F(.)$ is a convex and increasing function, e.g., $F(\boldsymbol{d}_s) = (\sum_{u\in\mathcal{U}(s)} d_u)/|\mathcal{U}(s)|$.

The objective function is the sum of the above costs, weighted by parameters α_1, α_2 and α_3, respectively. Note that including a delay cost in the objective function is for pushing conferencing delays experienced by users to be as small as possible, although we have constrained their upper bound by (8).

Problem formulation. Putting all pieces together, we cast the optimization problem as

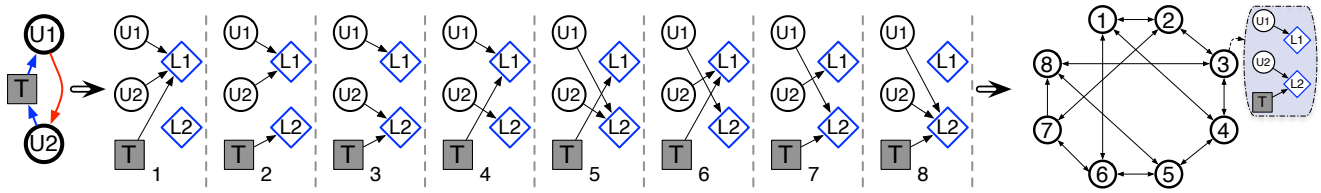
UAP: $$\min_{\lambda_{lu},\gamma_{lruv},\forall l,r,u,v} \quad \sum_{s\in\mathcal{S}}(\alpha_1 F(\boldsymbol{d}_s) + \alpha_2 G(\boldsymbol{x}_s) + \alpha_3 H(\boldsymbol{y}_s))$$

s.t. Constraints (1)-(8).

Remarks. Design parameters $\alpha_i \geq 0$ can be adjusted to achieve any desired performance/cost trade-off, e.g., larger α_1 leans more towards optimizing conferencing peformance, while larger α_2 and α_3 stress operational cost minimization. In Sec. IV, we will evaluate the impact of these parameters with experiments. Moreover, separability of the objective function across the sessions provides a nice opportunity to achieve a parallel algorithm, to be discussed next.

III. Algorithms and Discussion

We remark that tackling problem **UAP** even in a centralized manner is difficult, due to its combinatorial nature and persistent dynamics in the system.We prefer a parallel and adaptive solution —each session solves its assignment problem locally, such that the solution can scale with the problem size and adapts to the dynamics. Recently proposed Markov approximation approach [7] is one technique that allows us to construct one such solution. The overview of our solution approach is as follows. *First,* in Sec. III-A, we devise a Markov-based parallel and adaptive user-to-agent assignment algorithm that runs in one agent of each session (e.g., the session initiator's agent). The algorithm proceeds in an iterative fashion and converges to a near optimal assignment solution. The original Markov approach may suffer



(a) All 8 feasible assignment solutions, assuming both cloud agents are powerful enough and the end-to-end delays of both flows are always less than $D^{\max}$.

(b) The Markov chain

Fig. 4. A simple VC scenario with 1 session, 2 users, 1 transcoding operation, and 2 agents

slow convergence. *Second,* in Sec. III-B, we propose a fast bootstrapping algorithm which achieves a feasible close-to-optimal initial assignment.

A. Markov Approximation-Based Parallel Algorithm

Generally, the Markov approximation framework tackles combinatorial optimization problems in a decentralized manner by 1) constructing a class of problem-specific Markov chains with a target steady-state distribution and 2) investigating a particular structure of the Markov chain that is amenable to decentralized implementation.

1) Approximation Framework: Let $f = \{\boldsymbol{\lambda}, \boldsymbol{\gamma}\} \in \mathcal{F}$ be a feasible solution to problem **UAP**, where $\mathcal{F}$ is the set of all feasible solutions, i.e., all assignments that satisfy constraints (1)-(8). Let Φ_f denote the objective function value of problem **UAP** achieved by solution f and p_f denote the percentage of time that f should be in use. Using these notations we can rewrite problem **UAP** as follows:

$$\textbf{UAP-EQ:} \quad \min_{p_f, \forall f \in \mathcal{F}} \quad \sum_{f \in \mathcal{F}} p_f \Phi_f,$$
$$\text{s.t.} \quad \sum_{f \in \mathcal{F}} p_f = 1.$$

We can formulate the approximation version, **UAP**-β, of problem **UAP-EQ** using *log-sum-exp* approximation, where β is a large positive constant that controls the accuracy of the approximation [7]:

$$\textbf{UAP-}\beta: \quad \min_{p_f, \forall f \in \mathcal{F}} \quad \sum_{f \in \mathcal{F}} p_f \Phi_f + \frac{1}{\beta} \sum_{f \in \mathcal{F}} p_f \log p_f$$
$$\text{s.t.} \quad \sum_{f \in \mathcal{F}} p_f = 1.$$

UAP-β is a convex optimization problem. So we can solve its KKT conditions and derive its optimal solution

$$p_f^\star = \frac{\exp(-\beta \Phi_f)}{\sum_{f' \in \mathcal{F}} \exp(-\beta \Phi_{f'})}, \quad f \in \mathcal{F}, \tag{9}$$

and the optimal objective function value

$$\hat{\Phi} = -\frac{1}{\beta} \log \Big(\sum_{f \in \mathcal{F}} \exp(-\beta \Phi_f) \Big). \tag{10}$$

Moreover, the optimality gap between the optimal objective values of **UAP**-β and **UAP** is characterized by:

$$\min_{f \in \mathcal{F}} \Phi_f - \frac{1}{\beta} \log |\mathcal{F}| \le \hat{\Phi} \le \min_{f \in \mathcal{F}} \Phi_f. \tag{11}$$

Note that the approximation gap vanishes as β approaches infinity, i.e., the larger β is, the more accurate the approximation model is. We will further investigate the impact of β on the performance of our algorithm in Sec. IV.

The idea of introducing the above approximation framework is to approximate the optimal solution to problem **UAP** by time-sharing among its feasible solutions $f \in \mathcal{F}$ according to $p_f^\star$ in (9). Towards this, the key is to construct a Markov chain, which models feasible solutions as states, achieves stationary distribution $p_f^\star, \forall f \in \mathcal{F}$, and allows efficient parallel construction among the VC sessions.

2) Algorithm Design: Our parallel and adaptive algorithm pursues the near-optimal assignment solution to problem **UAP** by simulating such a Markov chain over time. Especially, the algorithm starts with a feasible assignment solution f of **UAP**, and may transit to another feasible solution f' according to the transition rate $q_{f,f'}$. The near-optimal solution is achieved when the Markov chain converges to the steady-state distribution $p_f^\star$ in (9). The transition rates should be carefully computed to achieve this steady-state distribution. In addition, though we have given the concrete form of $p_f^\star$ in (9), we should note that it is computed using KKT conditions in a centralized fashion, which requires complete, static information in the entire system. It adds a further challenge to compute the transition rates in a parallel fashion (in each session respectively), in order to achieve the desired overall stationary distribution.

Based on the theoretical insights from [7], the sufficient conditions in constructing such a Markov chain is to ensure that in the Markov chain: (i) any two states are reachable from each other (i.e., the Markov chain is irreducible); and (ii) the detailed balance equation is satisfied, $p_f^\star q_{f,f'} = p_{f'}^\star q_{f',f}, \forall f, f' \in \mathcal{F}$.

Sufficiency of these requirements is the key to provide a large degree of freedom in Markov chain design and leads to decentralized-friendly Markov implementation. The *first* degree of freedom is that we are allowed to set the transition rate between any two states to zero if they are still reachable from any other states. In this way, the stationary distribution of the modified Markov chain distribution is still $p_f^\star$.

On the other hand, direct transition between two states corresponds to migration of the system from one feasible assignment solution to another. To minimize the solution migration overhead, we allow direct links between two states in the Markov chain only if the value of exact one decision variable differs between the two corresponding assignment solutions. An example Markov chain is depicted in Fig. 4(b)

Algorithm 1: Markov approximation-based assignment (*for each session s*)

1 **procedure** WAIT
2 Generate an exponentially distributed random number with mean $\frac{1}{\tau}$ and begin countdown according to it
3 **while** *the timer has not expired*
4 **if** *Receive a* FREEZE *message* **then** Pause
5 **if** *Receive a* UNFREEZE *message* **then** Resume
6 **end**
7 Invoke HOP
8 **end procedure**
9 **procedure** HOP
10 Broadcast a FREEZE message to other sessions
11 Fetch the updated list of residual capacities of agents
12 $\mathcal{F}_s \leftarrow$ set of all feasible solutions with only one different decision
13 Migrate to solution $f' \in \mathcal{F}_s$ with probability proportional to $\exp(\frac{1}{2}\beta(\Phi_{s,f} - \Phi_{s,f'}))$
14 Broadcast a UNFREEZE message to other sessions
15 Invoke WAIT
16 **end procedure**

corresponding to the scenario in Fig. 4(a). Consider feasible solution 1 in Fig. 4(a) where both users and the transcoding task are assigned to L1, and feasible solution 2 where both users are assigned to L1 but the transcoding task is assigned to L2. They differ by only one assignment decision, so that there are direct links between state 1 and state 2 in Fig. 4(b).

Second, for two solutions (Markov chain states) f and f' with direct transitions, there are many options in designing transition rates $q_{f,f'}$ and $q_{f',f}$. To facilitate parallel Markov implementation, we design the transition rate between two states as

$$q_{f,f'} = \tau \exp\left(\frac{1}{2}\beta(\Phi_f - \Phi_{f'})\right) = \tau \exp\left(\frac{1}{2}\beta(\Phi_{s,f} - \Phi_{s,f'})\right),$$

where $\Phi_{s,f}$ and $\Phi_{s,f'}$ are the *local objective* values of session s (i.e., $\alpha_1 F(\boldsymbol{d}_s) + \alpha_2 G(\boldsymbol{x}_s) + \alpha_3 H(\boldsymbol{y}_s)$) at solutions f and f', respectively and τ is a positive constant. In our algorithm based on the value of τ each session initiates a timer and when timer expires the session chooses another assignment. The larger τ reduces the convergence of the algorithms, while it may impose the overhead of frequent assignment migration. The second equation above shows that we can calculate the transition rate using the local objective function values of the sessions, which enables parallel implementation of the algorithm. The rationale behind is that we allow only one decision variable's value to be different between f and f'. It is easy to show that this transition rate satisfies the detailed balance equations.

The procedures of our parallel algorithm are summarized in Alg. 1. First, we mention that the algorithm is executed in a representative agent of each session (e.g., the session initiator's agent). In HOP procedure, session s migrates to another feasible assignment with a probability proportional to the local objective value of the target solution, i.e., the lower the target objective value is, the more probable the session is to migration to it. Note that to compute the transition probability in Line 13, only the knowledge of local objective of the corresponding session is required, so the algorithm could be implemented in a fully parallel manner. In WAIT procedure, if the corresponding agent of session s receives a FREEZE message, it pauses its countdown, since another session is migrating, and resumes its countdown afterwards, which is still exponentially distributed because of the memoryless property of exponential distribution. This is the rationale that an exponential count down timer is used. We finally remark that the FREEZE message is passed as an intra-message within the cloud agents that operate in synchronized manner in a single cloud environment. The following proposition shows that independent of the initial assignment, Alg. 1 converges to the stationary state with provable convergence time (mixing time), with proof given in [7]

Proposition 1: Alg. 1 realizes a continuous-time Markov chain, which converges to the stationary distribution in Eq. (9).

We remark that in some similar approaches like simulated annealing [16], Gibbs sampling, and other Monte Carlo Markov chain approaches [6], the main idea is to sample a set of states according to a desired distribution by implementing a Markov chain that has the desired steady-state distribution. Hence, these approaches share the idea similar to Markov approximation. However, unlike Markov approximation, Gibbs sampling and simulated annealing do not explicitly consider designing the Markov chain in a way that it can be implemented in a distributed or parallel manner. As such, they cannot be directly leveraged to design parallel solutions desirable for our problem. In this paper, we follow Markov approximation framework to explicitly take into account the parallel implementation into Markov chain design, and tackle a unique set of challenges, such as Markov chain connectivity topology and transition rate design, to achieve Alg. 1. In addition, unlike the similar approaches that are incompetent against the system dynamics and noisy measurement of the problem data, Markov approximation framework can provide theoretical robustness to both system dynamics and noisy measurements, which is discussed in details in the next subsection.

3) Robustness to System Dynamics and Noisy Measurements: Our parallel algorithm is robust to variations due to session dynamics, i.e., addition and completion of a session. In the case that a new session starts, it can be bootstrapped with any feasible assignment solution, and then the agent which the session initiator is connecting to can execute its local algorithm by starting its countdown process.

In addition, our algorithm also adapts well to inaccurate measurements of the agent-to-user latencies and transcoding latencies, which lead to inaccurate values of $\Phi_{s,f}$ and $\Phi_{s,f'}$ and hence perturbed transition rates of each session s. The latency values between the users and the agents are *perturbed*, in practice. In addition, transcoding latency is highly dependent on both content characteristics and agents' load. Hence, the transition rate of each session is subject to the perturbation with inaccurate values of both $\Phi_{s,f}$ and $\Phi_{s,f'}$. Consequently, with perturbed values of objective function, Alg. 1 may converge to a sub-optimal steady-state distribution. Fortunately, our employed theoretical approach can provide a bound on the optimality gap due to the perturbation errors using a quantization error model.

We assume the perturbed Φ_f takes only one of the following discrete values $[\Phi_f - \Delta_f, \ldots, \Phi_f - \frac{1}{n_f}\Delta_f, \Phi_f, \Phi_f + \frac{1}{n_f}\Delta_f, \ldots, \Phi_f + \Delta_f]$ and the perturbed Φ_f takes the value $\Phi_f + j/n_f\Delta_f$ with probability $\eta_{j,f}$ and $\sum_{j=-n_f}^{n_f} \eta_{j,f} = 1$, where Δ_f is the error bound on configuration f and n_f is a positive constant.

Theorem 1: The stationary distribution of the perturbed assignment-hopping Markov chain is

$$\bar{p}_f = \frac{\delta_f \exp(-\beta\Phi_f)}{\sum_{f' \in \mathcal{F}} \delta_{f'} \exp(-\beta\Phi_{f'})}, \quad \forall f \in \mathcal{F}, \tag{12}$$

where $\delta_f = \sum_{j=-n_f}^{n_f} \eta_{j,f} \exp(\beta \frac{j\Delta_f}{n_f})$, and optimality gaps are

$$0 \leq \Phi^{\text{avg}} - \Phi^{\min} \leq \frac{(U+\theta^{\text{sum}}) \log L}{\beta}, \tag{13}$$

$$0 \leq \bar{\Phi}^{\text{avg}} - \Phi^{\min} \leq \frac{(U+\theta^{\text{sum}}) \log L}{\beta} + \Delta^{\max}, \tag{14}$$

where $\theta^{\text{sum}} = \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{U}} \theta_{uv}$ is the total number of transcoding tasks, $\Delta^{\max} = \max_{f \in \mathcal{F}} \Delta_f$ is the maximum perturbation error, $\Phi^{\min} = \min_{f \in \mathcal{F}} \Phi_f$ is the optimal value of **UAP**, $\Phi^{\text{avg}} = \sum_{f \in \mathcal{F}} p_f^\star \Phi_f$ is the expected objective with the original Markov chain, and $\bar{\Phi}^{\text{avg}} = \sum_{f \in \mathcal{F}} \bar{p}_f \Phi_f$ is the expected objective with the perturbed Markov chain.

The proof is relegated to Appendix A. Note that Eqs. (13) and (14) signify when β increases the optimality gap of the perturbed Markov chain decreases. But, the larger β values may increase the convergence time of Alg. 1 [21]. Moreover, the bounds are independent of the specific values of configurations, i.e., n_f and $\eta_{j,f}$.

B. AgRank Algorithm

We proceed to design an agent ranking algorithm for identifying a good starting feasible assignment solution, for bootstrapping the Markov approximation-based algorithm. The intuition is that if Alg. 1 can start from a close-to-optimal assignment, not only high-quality conferencing experience can be provided to the users starting from the beginning, but also fast convergence of the algorithm can be achieved.

In a nutshell of the algorithm which we refer to as *AgRank*, upon the start of a session, a potential agent of the session (e.g., the nearest agent to the session initiator) identifies a set of potential agents, ranks the agents, and assigns the users and transcoding tasks based on the ranking. Based on the example in Fig. 2, inter-agent delay is important in agent ranking, in addition to the agents' residual capacities and user-to-agent delay. The design of *AgRank* is motivated by the idea of Google's PageRank [4] and topology-aware node ranking in virtual network embedding [9] and is summarized in Alg. 2.

Constructing the potential agent list. In the first step, a set of top n^{ngbr} closest agents, $\mathcal{N}(u)$, for user u are picked as the possible agents and then the set of potential agents of the session, $\mathcal{N}(s)$, is constructed by putting together $\mathcal{N}(u)$ of all users (Lines 1-6). The parameter $n^{\text{ngbr}} \in [1, L]$ is the maximum number of potential agents for each user that could be set on a per-session or per-user basis. Setting $n^{\text{ngbr}} = 1$ yields the nearest assignment and $n^{\text{ngbr}} = L$ results in subscribing all users to the highest ranked agent.

Agent Ranking. The second step is to rank the potential agents based on a random walk model [4]. We define the initial

Algorithm 2: *AgRank* (for each session s)

1 $\mathcal{N}(u) \leftarrow \emptyset$ `// set of potential agents of user` u
2 $\mathcal{N}(s) \leftarrow \emptyset$ `// set of potential agents of session` s
3 **foreach** *user* $u \in \mathcal{U}(s)$ **do**
4 $\quad \mathcal{N}(u) \leftarrow$ top n^{ngbr} nearest agents to u in $\mathcal{L}$.
5 $\quad \mathcal{N}(s) \leftarrow \mathcal{N}(s) + \mathcal{N}(u)$
6 **end**
7 $\epsilon > 0, t \leftarrow 0$
8 Initialize $\pi_l[0] = \frac{\hat{u}_l + \hat{d}_l + \hat{t}_l + \hat{\sigma}_l}{\sum_{k \in \mathcal{L}} \hat{u}_k + \hat{d}_k + \hat{t}_k + \hat{\sigma}_k}, \quad l \in \mathcal{N}(s)$
$\quad$ `//` $\hat{u}_l, \hat{d}_l, \hat{t}_l,$ `and` $\hat{\sigma}_l$ `are the normalized residual quadruple of agent` l
9 **repeat**
10 $\quad \boldsymbol{\pi}^T[t+1] \leftarrow \boldsymbol{\pi}^T[t]\hat{D}$
11 $\quad \delta \leftarrow \|\boldsymbol{\pi}[t+1] - \boldsymbol{\pi}[t]\|$
12 $\quad t \leftarrow t + 1$
13 **until** $\delta < \epsilon$
14 $\boldsymbol{\pi}^\star \leftarrow \boldsymbol{\pi}[t]$
15 **foreach** *user* $u \in \mathcal{U}(s)$ **do**
16 $\quad$ Assign u to $l_u^{\text{sel}} \leftarrow \arg\max_{l \in \mathcal{N}(u)} \pi_l^\star$
17 **end**

ranking of agent $l \in \mathcal{N}(s)$ as in Line 8, based on the normalized residual quadruple of agent l. In this way, the initial ranking of the agents is aware of the resource availability of the potential agents which turns *AgRank* into a *resource-aware* algorithm. Let $\hat{D} = [\hat{D}_{lk}]_{|\mathcal{N}(s)| \times |\mathcal{N}(s)|}$ as normalized inter-agent delay matrix where $\hat{D}_{lk} = (\min_{l',k' \in \mathcal{N}(s)} D_{l'k'})/D_{lk}$, and $\boldsymbol{\pi} = [\pi_l]_{l \in \mathcal{L}}$ be the vector of agent ranks. The rank vector is updated iteratively with $\boldsymbol{\pi}^T[t+1] = \boldsymbol{\pi}^T[t]\hat{D}$, whose rationale is to capture inter-agent delay in ranking and find the optimal agent ranks (Lines 7-14). It has been shown that this iterative procedure converges very fast to a unique vector $\boldsymbol{\pi}^\star = [\pi_l^\star]_{l \in \mathcal{L}}$, as optimal agent ranks [4].

User and transcoding assignment. Next, user u is assigned to the highest ranked agent within the set $\mathcal{N}(u)$ (Line 16). For transcoding task assignment, we apply the rule of thumb that when there are at least two destinations with the same downstream representations for the outgoing flow of a particular user, assigning the respective transcoding task at the source agent is a good solution, whose transcoded stream can be served to more than one destination. One may imagine several other schemes for assigning the transcoding tasks, but here we are only seeking a good feasible one.

C. Discussion

Real-time assignment migration without user experience degradation. Alg. 1 converges to a bounded neighborhood of the optimal solution at the expense of imposing overhead to establish the new assignments. In each migration, a momentary interruption in conferencing experience might be happened as a consequence of switching the outgoing and the incoming traffics into the new cloud agent. To provide migration without user experience degradation, VC provider can keep both the new and the old assignments active during switching procedures by bearing some intermittent redundant transmissions. Moreover, exploiting segmentation-based transcoding approaches [12], transcoding migration can be

done by terminating the current segment and initiating the transcoding of the new segment in the new agent. We mention the implementation details in Sec. IV.

Complexity Analysis. At each iteration of Alg. 1, a representative agent of session s (e.g., the session initiator's agent) computes all feasible solutions with only one different decision with a time complexity of $O(|\mathcal{U}(s)|^2 L)$. We further note that to compute the transition probability in Line 13 of Alg. 1, it only needs to have the knowledge of local objective of the corresponding session, so the algorithm could be implemented in a fully parallel manner without requiring the global knowledge of the network. The iterative scheme in *AgRank* yields precision ϵ with the number of iterations proportional to $\max\{1, -\log \epsilon\}$ [4]. Constructing candidate agents, user assignment, and transcoding assignment takes a computation time of $O(|\mathcal{U}(s)|L \log L)$, $O(|\mathcal{U}(s)|)$ and $O(|\mathcal{U}(s)|^2)$, respectively.

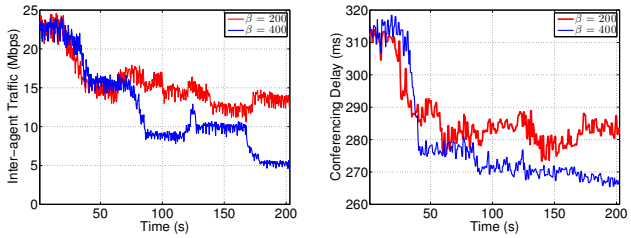
IV. Performance Evaluation

We evaluate the performance of our algorithms using: 1) a set of experiments based on prototype implementation of a real-world cloud-assisted conferencing system (Sec. IV-A), and 2) a set of large-scale trace-driven experiments (Sec. IV-B). We compare our solution to the *nearest* assignment policy (*Nrst*) (that is the assignment policy in *Airlift* [10] and *vSkyConf* [17]). For detailed illustration, we report the inter-agent traffic (corresponding to the operational cost) and the conferencing delay separately as the performance metrics, even though the objective is a weighted combination of them. As for the conferencing delay, we report the average delay of all users. For the end-to-end delay constraint (8), we set $D^{\max} = 400$ *ms* according to ITU-U G.114 [11].

A. Experiments on Prototype System

1) Prototype Overview and Setup.: We implement the cloud-assisted VC prototype software using the asynchronous networking paradigm in C++, and employ the OpenCV library [1] to capture video frames of device cameras in two representations and to transcode the streams. 6 Linux-based EC2 instances in different regions are employed as the cloud agents. A VC software is installed on them to execute our algorithms and to exchange and transcode the streams. Unless otherwise specified, we set the capacity of agents to be large enough and the transcoding latency of agents are in $[30, 60]$ ms, depending on the processing capabilities. Conferencing users are distributed in 10 locations (5 in North America, 4 in Asia, and 1 in Europe) using different operating systems. A lightweight conferencing software is installed on users that only transfers the video streams to/from an EC2 instance. We remark that the associated agent to the initiator of each session is responsible for executing Alg. 1 and *AgRank*, hence by migrating the execution of the algorithms to the cloud agents, no additional overhead is imposed to the client devices. Finally, we have launched 10 *actual* conferencing sessions, each with 3–5 participants.

We choose $\beta = 400$ in Alg. 1 which is proportional to the logarithm of the problem state space [7]. The countdown timer is set to 10 seconds, i.e., Alg. 1 executes every 10 seconds in each session on average. In each iteration, the assignment of either one user or one transcoding task is changed. When user-to-agent assignment migration is in progress, if we instantly tear down the old assignment, the other participants in the session experience streaming interruption (e.g., a frozen screen for a short period as 2-3 frames are delayed in a 30 fps video rate). We resolve such interruptions as follows: The migrated client sends its stream to both the old and the target agents for a short time interval (less than 30 ms on average according to the user-to-agent distances). This results in some overhead traffic that could be considered as the migration cost of the algorithm, whose volume (around 13.2 Kb corresponding to the 240p representation) is negligible as compared to the amount of traffic reduction after migration.



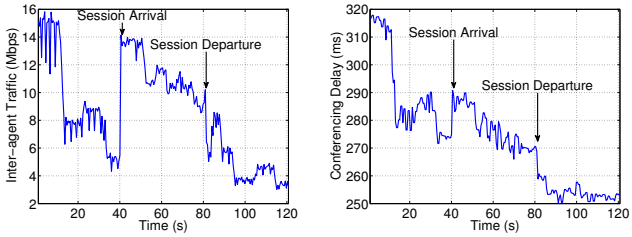
(a) Inter-agent traffic (b) Conferencing delay

Fig. 5. Evolution of traffic and delay over time (200 seconds) by executing Alg. 1 with different βs and *Nrst* for initial assignment

In Figs. 5-9, the initial cost/delay values at time 0 are results of either the *Nrst* or *AgRank* assignment policies, and running Alg. 1 following the initial assignment reduces them over time.

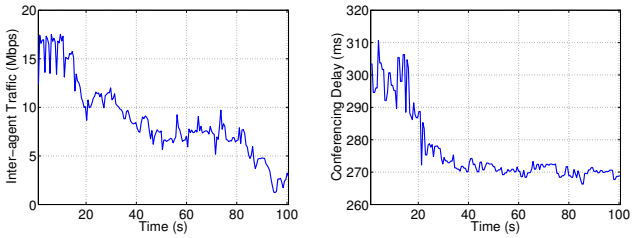
2) Traffic and Delay Reduction of Alg. 1: Fig. 5 demonstrates that Alg. 1 achieves significantly traffic and delay reduction, as compared to the initial assignment by *Nrst*, and converges in about 180 seconds. The fluctuations in the delay/traffic values are due to perturbations on actual data and assignment migrations. Comparing results of different βs in Fig. 5, we see that Alg. 1 with a lower value of β converges to the optimal assignment more slowly with higher fluctuations. In a dynamic scenario (Fig. 6), there are 6 sessions initially, 4 more sessions arrive at $t = 40$, and 3 sessions depart at $t = 80$. We can see that the algorithm adapts well to the dynamics by converging to new stable assignment solutions.

3) Effectiveness of AgRank: Comparing the initial traffic/cost values in Fig. 7 and Fig. 5, we can see that *AgRank* performs better than *Nrst* – 15 Mbps vs. 22 Mbps inter-agent traffic, with similar conferencing delays. In addition, starting from a close-to-optimal initial assignment by *AgRank*, Alg. 1 converges faster, i.e., obtained values at 100th second using *AgRank* for initial assignment are almost the same as those at 200th second with *Nrst*. We also note that although *AgRank* is an iterative algorithm, it is a fast algorithm, e.g., it takes less than 200 *ms* to find the optimal ranking of the agents upon session arrival on average in a micro EC2 instance. We finally remark that due to the parallel algorithm design, the convergence of the algorithm is independent of the number of users. In addition to *AgRank* that is a proper algorithm that reduces the convergence time of Alg. 1, the other candidate parameter to further reduce the convergence is the countdown parameter that may increase the migration overhead of the algorithm.



(a) Inter-agent traffic (b) Conferencing delay

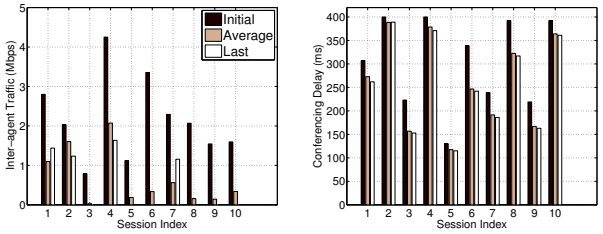
Fig. 6. Evolution of traffic and delay over time by executing Alg. 1 with $\beta = 400$ in the presence of session arrival/departure



(a) Inter-agent traffic (b) Conferencing delay

Fig. 7. Evolution of traffic and delay over time (100 seconds) by executing Alg. 1 with $\beta = 400$ and *AgRank* with $n^{\text{ngbr}} = 2$ for initial assignment

4) Case Study: While the previous figures show aggregate results in the entire system with 10 sessions, we study per-session results in Figs. 8-9. The initial assignments are obtained using the *Nrst* policy. First, per-session results are depicted in Fig. 8. Comparing values clearly shows that Alg. 1 reduces both inter-agent traffic and the average delay of users for all sessions. Second, in Fig. 9 we report the performance of 3 sample sessions in more details. For example, in session 8, 4 users subscribe to 3 different EC2 instances in Tokyo, Singapore, and Ireland initially, but soon all users are migrated to the Tokyo agent, resulting in zero inter-agent traffic. Due to the probabilistic nature of the system, a session may migrate to a worse assignment for some time, e.g., migration of session 9 at $t = 131$, but can recover soon, e.g., session 9 migrates back to the optimal assignment at $t = 141$.

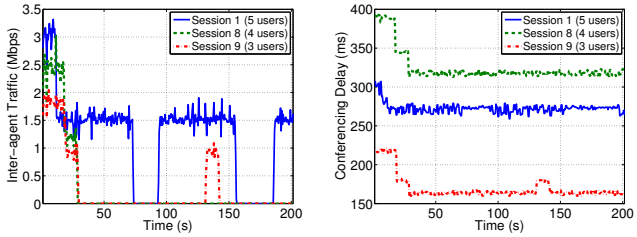


(a) Traffic per session (b) Delay per session

Fig. 8. Per-session improvements

B. Large-Scale Trace-Driven Experiments

1) Experimental Setup: We proceed to carry out Internet-scale experiments using 256 PlanetLab nodes as the users and 7 EC2 instances as the agents. We use the user-to-agent and inter-agent delays (approximately RTTs divided by 2) from [3], [18], where the RTTs are measured for 5 weeks at a granularity of one ping per second. 4 representations,



(a) Inter-agent traffic (b) Conferencing delay

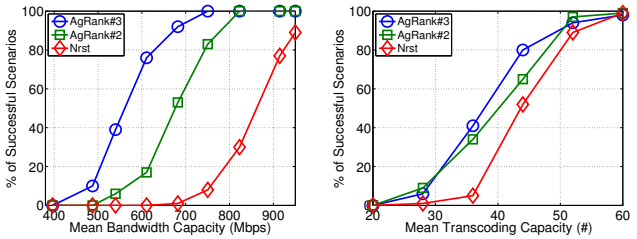
Fig. 9. Evolution of traffic and delay with Alg. 1 for the case of 3 sample sessions with different number of users

Alg.	Cost	Init.	Alg. 1		
			$\alpha_2 = 0$ (delay only)	$\alpha_1 = \alpha_2$	$\alpha_1 = 0$ (traffic only)
Nrst	Traffic	1443	979	829	521
	Delay	166	149	150	209
AgRank	Traffic	384	499	335	296
	Delay	176	162	163	214

TABLE II
THE IMPACT OF DESIGN PARAMETER α ON ALG. 1

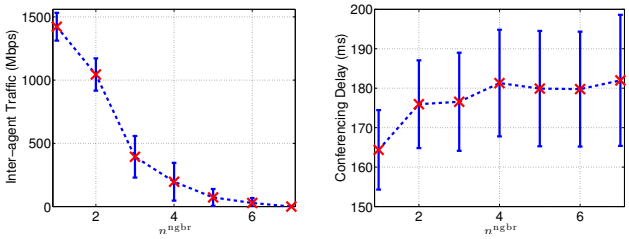
360p, 480p, 720p, and 1080p are exploited and a sparse transcoding matrix is considered such that 80% of users demand for 720p and only 20% demand for the others. The other parameter settings are the same as in the previous experiments, unless otherwise specified. In each experiment, we generate 100 random scenarios and plot the average results. In each scenario, there are 200 users in total (picked randomly from 256 PlanetLab nodes), who join different sessions, while each session has at most 5 users.

2) Impact of Design Parameters: The result is summarized in Table II. When $\alpha_1 = \alpha_2$, Alg. 1 using *Nrst* (*AgRank*) for initial assignment simultaneously reduces the traffic and delay from those of the commonly-adopted policy *Nrst* by 42% (77%) and 10% (2%), respectively. In addition, initialization by *AgRank* reduces the traffic by 73% at the expense of 6% longer conferencing delay in comparison with those in *Nrst*, while the longer delay could be compensated by Alg. 1. These observations corroborate our claim that the nearest policy yields neither minimal delay nor minimal operational cost, and our user-to-agent assignment design can significantly improve the conferencing experience and reduce the operational cost as a "win-win" solution for both the users and the conferencing provider. In addition, results in Table II clearly reveal that paying more attention to one part of the hybrid objective function may sacrifice the other. This justifies that the hybrid structure of the objective function is vital in design.



(a) different bandwidth capacities (b) different transcoding capacities

Fig. 10. Comparison of *AgRank* and *Nrst*



(a) Inter-agent traffic (b) Conferencing delay

Fig. 11. The impact of n^{ngbr} on *AgRank*

3) The Details of AgRank: The previous results showed that *AgRank* significantly outperforms *Nrst* by reducing the initial traffic cost. This reduction could be translated into an increased success rate of the initial assignment, i.e., all users in the system can successfully subscribe to agents, by serving more sessions with limited capacities of the agents. In Fig. 10, we show the percentage of successfully initialized scenarios (out of 100 random scenarios), with two versions of *AgRank*, *AgRank#2* with $n^{\mathrm{ngbr}} = 2$ and *AgRank#3* with $n^{\mathrm{ngbr}} = 3$, and *Nrst* under different average bandwidth capacities (Fig. 10(a), unlimited transcoding capacity) and transcoding capacities of the agents (Fig. 10(b), unlimited bandwidth capacity). We observe that with *AgRank#3*, all 100 random scenarios can be successfully initialized under average bandwidth capacity 750 Mbps, while with the resource-oblivious *Nrst*, only 8% of the randomly generated scenarios can be successfully initialized. The higher success rates of *AgRank*#3 than *AgRank*#2 show that picking among a larger number of potential agents provides a larger feasible set. To explore this further, we compare the performance of *AgRank* under different values of n^{ngbr} in Fig. 11. Clearly, $n^{\mathrm{ngbr}} = 1$, by which *AgRank* is equivalent to *Nrst*, yields the highest traffic cost. With $n^{\mathrm{ngbr}} = L$, all users of each session are subscribing to one agent and hence suffer from long conferencing delays.

V. Related Work

Before the upsurge of the cloud paradigm, P2P was deemed as an alternative to the client/server model. In [8], a P2P-based VC problem is tackled in utility maximization framework. However, the lack of powerful nodes in P2P hinders proper execution of high processing tasks. The idea of exploiting cloud bandwidth resources for VC is first proposed in *Airlift* [10]. Next, the authors in [17] employ the processing power of cloud for transcoding, in addition to the bandwidth resources. As mentioned before, these works adopt the *nearest* assignment policy which suffers from excessive resource usage. In very recent work [20], the authors propose a server placement and topology control approach to *only* minimize the latency in *transcoding-free* VC, without considering provider's cost.

Using the virtual network embedding paradigm [9] in [13], a primal-dual algorithm is proposed for resource allocation in real-time multimedia that could be customized to encompass video conferencing. Different from [13], here, deep study of problem **UAP** disclosed a difficult non-linear optimization problem that makes finding the solution using primal-dual approach incompetent. The idea of migration and re-optimizing the current configuration have been widely used in virtual networking problems for ameliorating the acceptance rate of vitrual networks [19], energy saving [15], etc. These goals could also be imagined as additional motivations of assignment migration in our problem.

VI. Conclusions

This paper addressed the cloud-assisted VC problem from the perspectives of user-to-agent assignment and transcoding task assignment, with the goal of designing a joint cost effective and low delay solution. Two successive algorithms are proposed: a decentralized algorithm to optimize the assignment tasks and a bootstrapping algorithm to achieve a close-to-optimal initial point for the former. Observations on extensive experiments corroborated our claim that user assignment is a critical design choice that results in a big difference in system performance. Experimental results demonstrated the superiority of our design compared to the existing work in terms of reduced delay and cost, and thus makes it as viable win-win solution for both the users and the VC service provider.

References

[1] http://opencv.org/.

[2] Cisco VNI service adoption forecast, 2012–2017. *White Paper, February*, 2013.

[3] P. Bailis, A. Davidson, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Highly Available Trans.: virtues and limitations. In *VLDB*, 2014.

[4] M. Bianchini, M. Gori, and F. Scarselli. Inside PageRank. *ACM Trans. on Int. Tech.*, 5(1):92–128, 2005.

[5] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[6] P. Bremaud. *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*, volume 31. springer, 1999.

[7] M. Chen, S. C. Liew, Z. Shao, and C. Kai. Markov approximation for combinatorial network optimization. *IEEE Trans. Inf. Theory*, 59(10):6301–6327, 2013.

[8] X. Chen, M. Chen, B. Li, Y. Zhao, Y. Wu, and J. Li. Celerity: A low-delay multi-party conferencing solution. In *ACM Multimedia*, pages 493–502, 2011.

[9] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang. Virtual network embedding through topology-aware node ranking. *ACM SIGCOMM Comp. Comm. Rev.*, 41(2):38–47, 2011.

[10] Y. Feng, B. Li, and B. Li. Airlift: Video conferencing as a cloud service using inter-datacenter networks. In *IEEE ICNP*, 2012.

[11] ITU-T. G. 114. *One-way transmission time*, 18, 2000.

[12] F. Jokhio, A. Ashraf, S. Lafond, I. Porres, and J. Lilius. Prediction-based dynamic resource allocation for video transcoding in cloud computing. In *IEEE PDP*, 2013.

[13] J. Liao, P. Chou, C. Yuan, Y. Hu, and W. Zhu. Online allocation of communication and computation resources for real-time multimedia services. *IEEE Trans. Multimedia*, 15(3):670–683, 2013.

[14] Y. Liu, F. Li, L. Guo, B. Shen, and S. Chen. A server's perspective of internet streaming delivery to mobile devices. In *IEEE INFOCOM*, pages 1332–1340, 2012.

[15] E. Rodriguez, G. Alkmim, D. Batista, and N. da Fonseca. Live migration in green virtualized networks. In *IEEE ICC*, pages 2262–2266, 2013.

[16] P. J. Van Laarhoven and E. H. Aarts. *Simulated annealing*. Springer, 1987.

[17] Y. Wu, C. Wu, B. Li, and F. C. Lau. vSkyConf: Cloud-assisted multi-party mobile video conferencing. In *ACM SIGCOMM Workshop on Mobile Cloud Computing*, pages 33–38, 2013.

[18] Z. Wu and H. V. Madhyastha. Understanding the latency benefits of multi-cloud webservice deployments. *ACM SIGCOMM Comp. Comm. Rev.*, 43(1):13–20, 2013.

[19] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. *ACM SIGCOMM Comp. Comm. Rev.*, 38(2):17–29, 2008.

[20] S. Zhang, D. Niu, Y. Hu, and F. Liu. Server selection and topology control for multi-party video conferences. In *ACM NOSSDAV*, 2014.

[21] S. Zhang, Z. Shao, M. Chen, and L. Jiang. Optimal distributed P2P streaming under node degree bounds. *IEEE/ACM Trans. Netw.*, 22(3), 2014.

APPENDIX

A. Proof of Theorem 1

To prove, first we should prove that the stationary distribution of the perturbed Markov chain is Eq. (12), this part is very similar to the proof in [21] and hence we proceed to prove Eqs. (13)-(14).

Let us define the dirac distribution as follows

$$\hat{p}_f = \begin{cases} 1 & \text{if } f = f^{\min}, \\ 0 & \text{otherwise.} \end{cases}$$

where $f^{\min}$ is the optimal solution of problem **UAP** (i.e., $f^{\min} = \arg\min_{f \in \mathcal{F}} \Phi_f$). In addition, $p_f^\star$ as defined in Eq. (9) is the optimal solution for problem **UAP**-β. Hence, using the result is Eq. 11 we have

$$\sum_{f \in \mathcal{F}} p_f^\star \Phi_f + \frac{1}{\beta} \sum_{f \in \mathcal{F}} p_f^\star \log p_f^\star \leq \sum_{f \in \mathcal{F}} \hat{p}_f \Phi_f + \frac{1}{\beta} \sum_{f \in \mathcal{F}} \hat{p}_f \log \hat{p}_f. \tag{15}$$

By Jensen's inequality [5] we get

$$-\sum_{f \in \mathcal{F}} p_f \log p_f = \sum_{f \in \mathcal{F}} p_f \log \frac{1}{p_f} \tag{16}$$

$$\leq \log \left(\sum_{f \in \mathcal{F}} p_f \frac{1}{p_f} \right) = \log |\mathcal{F}|. \tag{17}$$

Moreover, we have $\Phi^{\text{avg}} = \sum_{f \in \mathcal{F}} p_f^\star \Phi_f$ and $\Phi^{\min} = \sum_{f \in \mathcal{F}} \hat{p}_f \Phi_f$, by combining these equations we get

$$\Phi^{\text{avg}} \leq \Phi^{\min} + \frac{1}{\beta} \log |\mathcal{F}|. \tag{18}$$

We know that $|\mathcal{F}| \leq L^{U+\theta^{\text{sum}}}$, where U, θ^{sum}, and L are the total numbers of the users, the transcoding tasks, and the agents, respectively, hence

$$0 \leq \Phi^{\text{avg}} - \Phi^{\min} \leq \frac{(U + \theta^{\text{sum}}) \log L}{\beta}. \tag{19}$$

The above equation proves the inequality in Eq. (13).

In the next step, we prove the optimality gap of the perturbed Markov chain as characterized in Eq. (14). By reformulating Eq. (13) for the perturbed Markov chain we have

$$\sum_{f \in \mathcal{F}} \bar{p}_f \Phi'_f - \min_{f' \in \mathcal{F}} \Phi'_f \leq \frac{(U + \theta^{\text{sum}}) \log L}{\beta}, \tag{20}$$

where Φ'_f is the modified objective function of problem **UAP**-β in perturbed setting and is defined as $\Phi'_f = \Phi_f - \frac{\log \delta_f}{\beta}$, then by substituting the value of Φ'_f in Eq. (20) we get

$$\sum_{f \in \mathcal{F}} \bar{p}_f (\Phi_f - \frac{\log \delta_f}{\beta}) - \min_{f' \in \mathcal{F}} (\Phi_f - \frac{\log \delta_f}{\beta}) \leq \frac{(U + \theta^{\text{sum}}) \log L}{\beta}, \tag{21}$$

In addition, since $\Delta^{\max}$ is the maximum perturbation error, we have $\delta_f \leq \exp(\beta \Delta^{\max}), f \in \mathcal{F}$ and hence

$$\frac{\log \delta_f}{\beta} \leq \Delta^{\max}, \quad f \in \mathcal{F}. \tag{22}$$

Finally, we have $\bar{\Phi}^{\text{avg}} = \sum_{f \in \mathcal{F}} \bar{p}_f \Phi_f$ and then by combining Eq. (21) and Eq. (22), we get

$$0 \leq \bar{\Phi}^{\text{avg}} - \Phi^{\min} \leq \frac{(U + \theta^{\text{sum}}) \log L}{\beta} + \Delta^{\max}. \tag{23}$$

This proves the inequality in Eq. (14).