Cooperative Group Provisioning with Latency Guarantees in Multi-Cloud Deployments

Sean Yaw, Eben Howard, Brendan Mumey, Mike P. Wittie
Department of Computer Science, Montana State University
Bozeman, MT, USA
{sean.yaw, eben.howard, mumey, mwittie}@cs.montana.edu

ABSTRACT

Given a set of datacenters and groups of application clients, well-connected datacenters can be rented as traffic proxies to reduce client latency. Rental costs must be minimized while meeting the application specific latency needs. Here, we formally define the Cooperative Group Provisioning problem and show it is NP-hard to approximate within a constant factor. We introduce a novel greedy approach and demonstrate its promise through extensive simulation using real cloud network topology measurements and realistic client churn. We find that multi-cloud deployments dramatically increase the likelihood of meeting group latency thresholds with minimal cost increase compared to single-cloud deployments.

1. INTRODUCTION

Innovative Internet applications and services increasingly need low latency communication to deliver a responsive user experience. Augmented reality, online gaming, video chat, and real-time language translation are examples, where a small decline in responsiveness is noticeable to users and can eventually lead to a drop in usage, resulting in loss of application revenue [12, 18, 20, 31, 40, 48]. Many of these applications include a social component, which allows users to *cooperate* in a shared online experience. Recent work shows the impact of one user's *lag* on the Quality of Experience of all group members in a cooperative game and argues the need to meet latency requirements for all members [33].

Network latency, a significant component of overall lag, is difficult to control when it stems from conditions in the public Internet, such as congestion and circuitous forwarding routes [22, 30]. For example, public network switching overhead comprises 38% of end-to-end latency [30]. Large-scale application providers reduce the public Internet's latency for their users by distributing private resources (e.g. traffic proxies, VPNs) closer to users and forwarding application traffic within private networks [22]. By contrast, small-scale application providers do not have access to the same scale of private infrastructure and remain at a disadvantage. This disparity makes it challenging for small providers to deliver a comparatively responsive and affordable application experience for all their users.

Despite this disadvantage, small-scale providers can leverage the increasingly geographically dispersed public cloud datacenters to reduce user request delay. Cloud datacenters are interconnected with low-latency Internet paths and are often located with large population centers. Small-scale application providers can employ cloud resources in multi-cloud deployments to act as a proxy network, thereby leveraging its low-latency properties [34, 41]. These low latency paths introduce triangle inequality violations which can be used in place of default routing paths to reduce end-to-end delay. Figure 1 shows an example network, in which proxies reduce path latency. Previous research has demonstrated how public cloud proxy networks reduce latency between last-mile networks and distant application datacenters [30, 51]. Large cloud providers such as Amazon (AWS) and Microsoft (Azure) already offer proxy products, but only within their networks [2, 10].

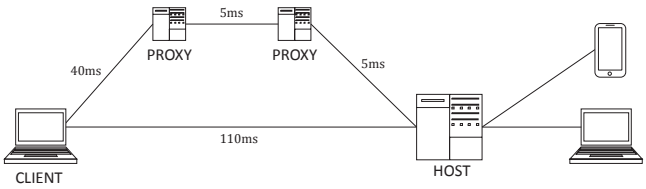


Figure 1: Proxy network used to decrease end-to-end latency for application client.

In this paper we address two challenges of cloud proxy networks:

First, can small-scale application providers deploy multi-cloud proxy networks in a cost effective manner? Larger geographical proxy distribution reduces average client path latency, but also increases cloud rental costs. Further, inter-datacenter traffic is more expensive when the two datacenters do not belong to the same cloud provider. Despite the potential for higher costs of widely distributed deployments, we demonstrate an attractive tradeoff between latency and cost, even in multi-cloud deployments.

Second, how can small-scale application providers allocate cloud resources to meet latency constraints of cooperative group applications? We propose a method to manage cloud resources (servers and communication links) to minimize their rental cost subject to a group latency threshold. Our model considers link and forwarding latency as part of total latency. We formally define the Cooperative Group Provisioning (CGP) problem, show that it is NP-hard to approximate within a constant ratio, and propose an algorithm with good performance on multiple simulated cloud provider networks.

Our results enable small-scale providers to incorporate multi-cloud proxy networks as one of the tools in the design of responsive distributed systems. Our algorithms offer developers a level of currently unavailable control over end-to-end network performance and will allow them to design co-

operative group applications with explicit latency/cost budgets, or predictable performance and cost tradeoffs.

The rest of this paper is organized as follows. We discuss related work in Section 2 and the system model in Section 3. Section 4 introduces the CGP problem and its complexity analysis. A greedy approach and considerations for group churn are presented in Section 5. Experimental results are presented in Section 6 and we conclude in Section 7.

2. RELATED WORK

Multi-cloud deployments are a rapidly maturing research area. A growing number of tools make it possible for developers to build applications that simultaneously use geographically distributed resources from several cloud providers. We present the state of scheduling tools and algorithms in that space to help frame the CGP problem.

2.1 Distributed Resource Management

The continued expansion of cloud computing creates opportunities for widely distributed application deployments. At the same time, standardization of cloud interfaces and advances in multi-datacenter and multi-cloud application deployments make it increasingly feasible for small scale providers to build dynamic proxy networks that adjust to application demands and network conditions.

Overlay networks have been explored to manage Internet traffic, with the goal of decreased loss rate [17] and reduced latency [45]. Our contributions differ from traditional overlay networks in that the network we propose (public multi-clouds) incur rental costs, and resources are chosen based on aggregate utilization, not on a client-by-client basis. We show the use of multiple public cloud providers has the potential to lower the latency between users and the host server for group applications. We also propose an approach to mitigate the extra costs of multicloud deployments, to make such deployments practical.

Large-scale providers developed tools (e.g. Borg, Mesos, Kubernetes, Omega) for application resource management within and across their datacenters [32, 37, 46]. Container technologies (e.g. Docker, ZeroVM) complement resource management frameworks to simplify the packaging of application code and OS dependencies to make code portable across heterogeneous cloud resources [7,36]. These technologies make it possible for small-scale application providers to achieve both agile vertical and horizontal resource scaling in multi-cloud deployments.

These solutions, however, do not advise developers on where and when to instantiate virtual resources. Conductor considers price, performance, as well as cloud service transfer costs to schedule MapReduce jobs [50]. Meryn considers computation time and throughput to meet job QoS constrains in platform-as-a-service (PaaS) clouds [25]. Quasar rectifies resource underutilization in Mesos and allows jobs to specify performance goals, notably in terms of latency [24]. Finally, Sparrow and alsched use a queue balancing method to schedule small jobs for fast completion [40]. These tools are not immediately suitable for multi-cloud deployments, because they do not take into account the cost and network performance between clients and datacenters. CloudTalk considers application traffic patterns and computes the cost of various placement strategies, but falls short of recommending a deployment strategy [43].

Recent multi-cloud deployment work does not address the costs and communication latency between clients and datacenters. Paraiso *et al.* describe a Multi-Cloud-PaaS (MCP) architecture to manage elasticity across multiple providers as a backstop to cloud outages [41]. Keahey *et al.* propose a system that provides similar elasticity for the software-as-a-service (SaaS) model in response to application-specific indicators [34]. Their model considers high network delay as an indicator of failure, but does not support latency-based deployment optimization. Work by Franceschelli *et al.* provides cost and performance estimates for multi-cloud deployments based on service layer objectives [28].

Finally, several works provide portals for multi-cloud resource management. Alrokayan and Buyya propose Cloud Web Portal (CWP) as a unified interface to manage VMs distributed across multiple clouds [15]. CWP scheduler takes network latency into account, but only in the context of batch task distribution, for example rendering jobs. Similar approaches for management of multi-cloud resources include Delta Cloud, jClouds, and CloudFoundry [4, 6, 8]. Commercial services in that space include RightScale, enStratus, and Kaavo [5, 9, 11]. These architecture are extendible and could present a useful platform for the deployment of the algorithms presented in this paper.

2.2 Related Optimization Problems

The systems solutions discussed above address the scheduling problem in the cloud, but do not take into account delay between clients and datacenters or group latency constraints. The problem of selecting cost effective, latency constrained paths through a network is similar to the well studied problem of Constrained Shortest Path (CSP).

In CSP, the single source/destination cheapest path, subject to a latency bound, is sought. Bernstein introduced a near linear time PTAS for undirected graphs that approximates both the path cost and latency [19]. Heuristics have been developed that have been found to be very efficient in practice [21]. CSP, however, does not have node weights to represent datacenter rental costs. Further, datacenter rental costs are shared by multiple clients, and CSP only considers single source/destination paths which doesn't provide the ability to capture shared costs.

The Constrained Steiner Tree (CST) problem, where the cheapest Steiner tree is found subject to a latency threshold, does share costs among multiple source nodes. Heuristics for the CST problem have been extensively studied since the early 1990's [35]. Modern approaches include genetic algorithms and ant colony optimization [26, 29]. CST also lacks node costs, restricts solutions to trees, and shares edge costs, which is not a realistic model for cloud networks.

Various efforts consider cloud resource allocation algorithms that respond to network conditions [13, 14]. These approaches do not include multiple groups of clients, nor do they consider application specific latency thresholds.

3. SYSTEM MODEL

We consider a scenario where groups of clients look to connect to their other group members through a cooperative application (e.g. a group of friends playing an online game and a group of colleagues engaged in a video conference). Each group, and thus each client, has an application specific latency threshold, as well as a datacenter (DC) serving as the group's host. DCs exist in the network that can

forward client traffic, in addition to application hosting.

These clients and DCs are all interconnected with links that have an associated usage cost and latency. Each DC has the capability to rent out virtual machines (VMs) that forward traffic. These VMs have an associated rental cost and a capacity, given in terms of the number of clients they are able to service simultaneously.

This network can be modeled as a graph, $G = (V, E)$, where the DCs and clients are nodes in the graph ($V = D \bigcup C$), and the links are edges. At each DC d, there is a VM rental cost $r(d)$, as well as a VM forwarding capacity $f(d)$. Each edge e has a rental cost $r(e)$ and latency $l(e)$, which includes the latency induced at the incident VMs. Each group (and thus, each client) has a hosting DC, $h_c \in D$, as well as a latency threshold, L_c. Many of these parameters can be time varying and our solution will use the current values at time of computation.

We make the following assumptions that reflect a broad range of network application architectures: 1) Each group's hosting DC, h_c, is determined ahead of time and provided when the group is formed. 2) A VM can forward traffic for clients in different groups. 3) VM costs are accrued by number purchased (i.e. $r(d)$ will be charged to cover any number of clients less than $f(d)$). 4) Link costs are accrued by the number of clients using the link (i.e. $r(e)$ will be charged each time e belongs to any client's path).

For any path p_c from client c to its host h_c, the rental cost of c's connection is,

$$r(p_c) = \sum_{e \in p_c} r(e)$$

Likewise, the latency of c's connection via path p_c is,

$$l(p_c) = \sum_{e \in p_c} l(e) \quad (1)$$

Given a path p_c for each client c, the number of VMs that need to be rented at DC d is,

$$v_d = \left\lceil \frac{|\{c : d \in p_c\}|}{f(d)} \right\rceil$$

Finally, the total rental cost for deployment, given a path p_c for each client c, is the sum of the clients' connection costs plus the cost of each VM instance:

$$r = \sum_{c \in C} r(p_c) + \sum_{d \in D} v_d \cdot r(d) \quad (2)$$

4. CGP FORMALIZATION

In this section we formalize the CGP problem and show it is NP-hard to approximate within a constant factor.

DEFINITION 1. *The input for the **Cooperative Group Provisioning (CGP)** problem is a set of clients nodes C, a set of DC nodes D (with cost and capacity values), links between nodes (with cost and latency values), an assignment of a host DC, h_c, for each client, as well as a latency threshold, L_c, for each client. The output is a path from each client to its host such that the latency of each path (equation 1) is below the latency threshold L_c and the total rental cost (equation 2) is minimized.*

THEOREM 1. *There exists a constant, $0 < \gamma < 1$, such that the CGP problem has no $\gamma \ln m$-approximation, where m is the number of clients, unless $P = NP$.*

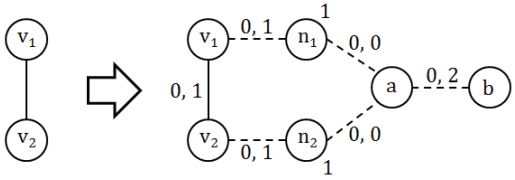


Figure 2: DOMINATING-SET reduction to CGP.

PROOF. This result is via an approximation-preserving reduction from the DOMINATING-SET problem: Given a graph $G = (V, E)$, find a minimal $U \subseteq V$ such that for all $v \in V \setminus U$, there is a $u \in U$ such that $(u, v) \in E$.

Let $G = (V, E)$ be an instance of DOMINATING-SET where $|V| = m$. Reduce G to a CGP instance, G', as follows: Make each node in V have cost 0 and edge in E have weight = (cost, latency) of $(0, 1)$. Add new nodes a and b with cost 0 and connect them with an edge having weight of $(0, 2)$. Create a set of new nodes, N, by adding a new node n_i with cost 1 for each node $v_i \in V$. For each n_i, add an edge that connects to v_i with weight of $(0, 1)$, and add an edge that connects to a with weight of $(0, 0)$. Figure 2 shows the reduction from DOMINATING-SET to CGP.

Let the set of clients $C = V \cup b$, the latency threshold for all clients be 2, and every node in G' have an unlimited forwarding capacity. A solution to G' is a subset of N, rented to connect all members of C to a. Node a must be the host for any valid solution to G' because $b \in C$, and a is the only node reachable within the latency bound.

Suppose that $U = \{v_i, ..., v_j\}$ is a dominating set of G. Since a is the host, every element of U can be reached with a latency of 1. Since every other element in $V \setminus U$ can be reached in one more (latency of 1) hop, every client can be reached with at most a latency of 2. Thus, $\{n_i, ..., n_j\}$ are the only nodes from N needed to form a solution to G'.

Suppose there is a solution to G', $\{n_i, ..., n_j\} \subseteq N$. Since n_k corresponds to an element of V, the subset $\{n_i, ..., n_j\}$ represents a unique subset, $S = \{v_i, ..., v_j\} \subseteq V$. Since the latency of each path is at most 2, each element of V must be in S (latency of 1) or directly connected to a member of S (latency of 2), making S a dominating set.

Since elements of N and V are in one-to-one correspondence, the number of nodes selected from N is the same as the number of nodes in the dominating set, thereby making the costs equal and the reduction approximation-preserving. It was shown in [44] that there exists a constant, $0 < \gamma < 1$, such that the DOMINATING-SET cannot be approximated within a factor of $\gamma \ln m$ unless $P = NP$, thus this result holds for CGP as well. □

5. A GREEDY ALGORITHM

Due to the complexity of the CGP problem, we examine a greedy strategy for the problem. Our approach is to consider multiple paths from each client to its host and select the path that will incur the least additional cost, considering the current state of the network (i.e. what VMs are already rented).

The algorithm first generates the k shortest paths (based on latency) from each client to its host and discards any paths that are over the latency threshold. For each path, the path's cost is determined to be the sum of the link costs plus the forwarding cost incurred at each intermediate DC. This forwarding cost depends on the number of forwarding

Algorithm 1 Greedy-k

Step 1 Generate the k shortest (by latency) paths for each client and put them in $\mathcal{P}$ if they are below the client's latency threshold.

Step 2 For each path in $\mathcal{P}$, determine its cost by summing all of its link costs and the cost to forward an additional client at the VMs along the path.

Step 3 Schedule the least cost path in $\mathcal{P}$, remove all other paths from $\mathcal{P}$ for that client, and update the VM forwarding numbers along the scheduled path.

Step 4 If $\mathcal{P}$ is not empty (i.e. client(s) still need to be scheduled), loop back to Step 2.

VMs currently owned at each intermediate DC and the capacity remaining on each VM. The paths are then sorted based on the total cost. The path with the least total cost is selected and added to the solution. Any VM purchases that need to be made to support this path are recorded. The process then loops back and determines the path costs for the paths of the remaining clients. This algorithm is presented as Algorithm 1.

A variety of methods could be used to generate the k shortest paths in Step 1 of the algorithm. For our evaluation in Section 6, we chose to use Yen's algorithm for finding the k shortest loopless paths [52]. Other shortest path algorithms could be used, or determining paths based on parameters other than latency, such as total link cost, are possible.

5.1 Group Churn

Any functional solution to the CGP problem must address group churn. Instances will not remain static, as groups come online and go offline routinely. We present a modification to Algorithm 1 that accounts for churn. This algorithm relies on the same underlying principle as Algorithm 1 by scheduling the cheapest path each iteration, until all clients have an assigned path. Churn is handled by only scheduling paths to new clients. Legacy clients retain the initially assigned paths.

When a new group comes online, each client is assigned a path that it will use for the rest of its connection. As such, for each moment in time, only the new clients need to be scheduled and they are done so by Algorithm 1 with consideration given to the already rented VMs. This algorithm is presented as Algorithm 2.

5.2 Multicast Considerations

The algorithms described so far select paths for unicast traffic. Some applications, such as video conferencing, send identical traffic to all group members. It is cost effective to distribute this traffic in a multicast fashion. Specifically, we describe a way to use Algorithm 1 to determine multicast traffic paths.

Algorithm 1 does not need to be modified to support multicast distribution. Instead, the input graph is modified. Existing nodes and edges between clients and DCs are not changed. New intermediate nodes are added to all inter-DC links. The cost of VMs on the new nodes is the cost of the link it is added to and the forwarding capacity is infinite. The link now split in two has no cost and one of the sublinks has the old latency value.

Algorithm 2 Greedy-k Churn

Step 1 Let the solution for timeslot, t, start as the solution to $t-1$ (empty if $t=0$) and let $\mathcal{C}$ be the set of new clients for t.

Step 2 Remove offline clients from the solution.

Step 3 Execute Algorithm 1 on the set $\mathcal{C}$.

Step 4 Add the solution from Algorithm 1 to the current solution and return to Step 1 to schedule a new timeslot.

Since VM costs are shared by all clients being serviced, we merely push the link cost to a VM on a new node. Then, when unicast paths are determined, they will leverage the existence of other traffic in the model (by way of already rented VMs). This enables link costs to be shared when identical data is transferred from host to clients.

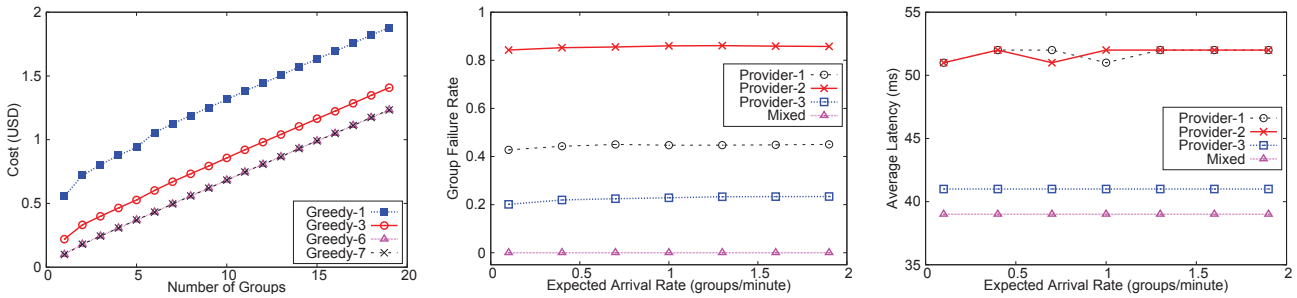
6. EXPERIMENTAL RESULTS

In this section we evaluate the effectiveness of our greedy algorithm from Section 5 in meeting the QoS needs of cooperative group applications in multi-cloud deployments. We base our evaluation on an extensive set of simulations on networks configured with connectivity and pricing models of Amazon, Microsoft, and Rackspace cloud providers. To avoid comparisons, we anonymize the identity of these services in the presented results.

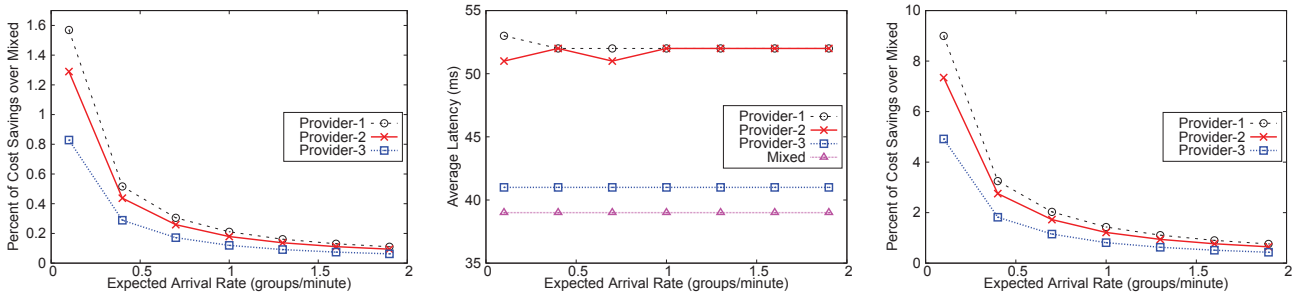
The simulation model used has been parametrized to reflect real-world connectivity between datacenters, as well as between datacenters and users in last-mile networks. We estimate the latency between datacenters as their great-circle (orthodrome) distance divided by half the speed of light to account for switching delay and coiled fiber in fiber optic conduits [49]. Latency between two datacenters that belong to the same cloud provider is assumed to be lower, because traffic traverses VPNs, rather than the public Internet. As such we only divide the distance of those links by two-thirds the speed of light. We assume the throughput of inter-datacenter links to be infinite, as their capacity is orders of magnitude larger than simulated traffic flows.

Server and bandwidth costs were taken from the advertised parameters of three top cloud service providers in the US [16,38,39,42]. The link pricing scheme in our models reflects the lower cost of traffic between two datacenters that belong to the same provider.

We randomly sample user locations from the locations and relative population sizes of the 100 most populous US cities. The latency between these clients and datacenters is based on distance plus last mile latency of access networks in a user's city based on FCC data [27]. These values contain the effect of low bandwidth and traffic congestion in access networks. All latency values include the latency induced by the VMs and were corroborated through direct ping measurements from cloud providers to random hosts (location verified with IP to location database) in last-mile networks of the 100 most populous US cities. To randomize the latency of different hosts from the same city, we assume our latency measurement represents the median latency in that



(a) k values: cost vs. num groups (b) Game churn: failure vs. arrival rate (c) Game churn: latency vs. arrival rate



(d) Game churn: cost vs. arrival rate (e) Video churn: latency vs. arrival rate (f) Video churn: cost vs. arrival rate

Figure 3: Performance of greedy algorithm for static instances, and churn of game and video chat groups.

city, which we then increase, or decrease based on the distribution of FCC measurements of latency for different link access speeds.

We simulate groups of clients utilizing two types of applications. The first is a gaming application with a latency threshold of 80 ms, group size of 12, and fixed session duration of 10 minutes [3,23]. The second is a video chat application with a latency threshold of 200 ms, group size of 10, and an exponentially distributed duration [1,47]. To calculate VM proxy capacity for these applications, we empirically evaluated the number of sessions that can be proxied by a VM on each data center without queuing request traffic. To calculate the traffic costs we measured the mean volume of flows in these applications using `pcap` captures.

Finally, we performed simulations on gaming groups without any churn to determine the optimal number of paths per client for which to configure Algorithm 1. The simulation was the average of ten iterations on instances of between one and twenty groups. The number of paths considered was one, three, six, and seven. Results of this simulation are shown in Figure 3(a). Using three paths results in an average cost savings of 40% compared to using only one path. Using six or seven paths resulted in an average cost savings of 50% over a single path, and diminishing marginal benefits over three paths. We decided to run the churn simulations with the number of paths for Algorithm 1 set at three.

6.1 Group Churn Evaluation

This section details the performance of Algorithm 2 with varying degrees of group churn and different cloud service provider networks over a 24 hour period. Group churn is modeled as group arrival times fitting a Poisson process. The degree of churn is manipulated by the expected arrival rate of groups, in groups per minute. Group duration is subject to the parameters discussed earlier.

We considered four different cloud solutions for group provisioning. Provider-1, 2, and 3 are the three top providers in the US. When Provider-n is referenced, only that provider's network can be used. The fourth solution, Mixed, represents a multi-cloud deployment on Providers 1, 2, and 3.

Figure 3(b) shows failure rate of groups meeting their latency threshold (80 ms for gaming) as a function of group arrival rate. The Mixed network is the only one that had a zero percent failure rate. Since the greedy algorithm determines shortest paths based on latency, it is guaranteed to find a path under the latency threshold, should one exist. Thus, the failure rate is not artificially high due to peculiarities of Algorithm 2. The decrease in failure rate is related to the number of DCs in the network. More geographically distributed multi-cloud deployments provide more opportunity for proxies to meet latency thresholds. Figure 3(b) also indicates the advantage to employing proxies. In our model, clients are able to connect to an application host in a cloud DC without using proxies. Thus, any failures seen when using proxies would also be failures without proxies.

Figure 3(c) shows the average client latency for increasing churn rates. As expected from the failure rate results, the networks that experience higher failure rates also have higher average latencies. The curves are mostly flat, suggesting that the average latency for each provider is more dependent on the location of datacenters than it is on the number of active groups. The independence of latency from group churn is due to the fact that even a small number of active groups leads to VMs being rented at most datacenter locations, thereby keeping the average latency stable.

Figure 3(d) shows the percent of cost savings of Provider-1, 2, and 3 compared to the Mixed network over a 24 hour period with increasing churn rates. The cost savings of using a single-cloud deployment starts at at most 1.6% and decreases with increasing churn rate, which points to profitability of multi-cloud deployments. The small variability of cost savings is due to pricing similarities between providers,

geographic proximity of DCs from different networks, and consistent selection of similar resources. This result suggests that for nearly the same cost, latency thresholds can be met with much greater reliability by utilizing multi-cloud deployments.

Results of churn simulations on video chat groups are similar to the results for game groups. Failure rate was zero for all networks due to the high latency threshold of 200 ms. Even without any latency failures on the single provider networks, Figure 3(e) shows that the average latency is still lower with a multi-cloud deployment. Further, Figure 3(f) shows that the cost savings of using a single-cloud deployment is low and decreases with the increasing rate of churn.

7. CONCLUSIONS

We introduced the CGP problem and showed it is NP-hard to approximate within a constant factor. We presented a versatile greedy approach that is applicable for static instances, realistic group churn, and applications that are able to leverage multicast traffic. This approach has shown to be efficient in realistic simulations and has the guarantee of finding feasible solutions if they exist.

Simulations on current US cloud providers indicate that multi-cloud deployments are cost effective in reducing latency. Larger numbers of DC locations lead to greater selection of resources to rent, which reduces average latency and group failure rate (tail latency). Our simulations have shown that leveraging these larger cloud networks can be expected to not adversely affect the cost of application deployment.

8. REFERENCES

[1] ITU-T G.114. http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-G.114-200305-I!!PDF-E, May 2003.

[2] AWS Direct Connect. http://aws.amazon.com/directconnect/, Dec. 2014.

[3] Call of Duty: Ghosts - Multiplayer. http://www.ign.com/wikis/call-of-duty-ghosts/Multiplayer, Apr. 2014.

[4] Cloud Foundry. http://cloudfoundry.org/, Dec. 2014.

[5] Dell Cloud Manager. http://enstratus.com/, Dec. 2014.

[6] δ-cloud. http://deltacloud.apache.org/, Dec. 2014.

[7] Docker. http://www.docker.com/, Dec. 2014.

[8] jClouds. http://jclouds.apache.org/, Dec. 2014.

[9] Kaavo. http://kaavo.com/, Dec. 2014.

[10] Microsoft Azure ExpressRoute. http://azure.microsoft.com/en-us/services/expressroute/, Dec. 2014.

[11] RightScale. http://rightscale.com/, Dec. 2014.

[12] M. Abrash. Latency the sine qua non of AR and VR. http://blogs.valvesoftware.com/abrash/latency-the-sine-qua-non-of-ar-and-vr/, Dec. 2012.

[13] M. Alicherry and T. V. Lakshman. Network aware resource allocation in distributed clouds. In *IEEE Infocom*, Mar. 2012.

[14] F. AlQayedi, K. Salah, and M. Zemerly. Network-aware resource allocation for cloud elastic applications. In *Electronics, Circuits, and Systems (ICECS)*, Dec. 2013.

[15] M. Alrokayan and R. Buyya. A Web portal for management of Aneka-based MultiCloud environments. In *Australasian Symposium on Parallel and Distributed Computing*, Jan. 2013.

[16] Amazon. Amazon EC2 pricing. http://aws.amazon.com/ec2/pricing/, July 2014.

[17] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *ACM SOSP*, Oct. 2001.

[18] N. Beighton. The human cloud: Wearable technologys impact on society. http://www.rackspace.com/blog/the-human-cloud-wearable-technologys-impact-on-society/, June 2014.

[19] A. Bernstein. Near linear time $(1+\epsilon)$-approximation for restricted shortest paths in undirected graphs. In *ACM-SIAM SODA*, Jan. 2012.

[20] K.-T. Chen, P. Huang, and C.-L. Lei. Effect of network quality on player departure behavior in online games. *Parallel Distributed Systems*, 20:593–606, May 2009.

[21] S. Chen, M. Song, and S. Sahni. Two techniques for fast computation of constrained shortest paths. *IEEE/ACM ToN*, 16(1):105–115, Feb. 2008.

[22] Y. Chen, S. Jain, V. K. Adhikari, and Z.-L. Zhang. Characterizing roles of front-end servers in end-to-end performance of dynamic content distribution. In *ACM IMC*, Nov. 2011.

[23] M. Claypool and K. Claypool. Latency and player actions in online games. *CACM*, 49(11):40–45, Nov. 2006.

[24] C. Delimitrou and C. Kozyrakis. Quasar: resource-efficient and QoS-aware cluster management. In *ASPLOS*, Mar. 2014.

[25] D. Dib, N. Parlavantzas, and C. Morin. Meryn: Open, SLA-driven, Cloud Bursting PaaS. In *ACM Workshop on Optimization Techniques for Resources Management in Clouds*, June 2013.

[26] R. Fabregat, Y. Donoso, F. Solano, and J. Marzo. Multitree routing for multicast flows: A genetic algorithm approach. In *Catalan Conference on Artificial Intelligence*, Oct. 2004.

[27] FCC. Measuring Broadband America - 2014. http://www.fcc.gov/reports/measuring-broadband-america-2014, June 2014.

[28] D. Franceschelli, D. Ardagna, M. Ciavotta, and E. Di Nitto. Space4cloud: A tool for system performance and costevaluation of cloud systems. In *Workshop on Multi-cloud Applications and Federated Clouds*, Apr. 2013.

[29] K. Ghoseiri and B. Nadjari. An ant colony optimization algorithm for the bi-objective shortest path problem. *Applied Soft Computing*, 10(4):1237–1246, 2010.

[30] I. Grigorik. Latency: The new Web performance bottleneck. http://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck/, July 2012.

[31] O. Hansen. The biggest problem in Augmented Reality: Latency. https://identifeye.wordpress.com/2013/01/03/the-biggest-problem-in-augmented-reality-latency/, Jan. 2013.

[32] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *USENIX NSDI*, Apr. 2011.

[33] E. Howard, C. Cooper, M. P. Wittie, S. Swinford, and Q. Yang. Cascading impact of lag on user experience in multiplayer games. In *ACM NetGames*, Dec. 2014.

[34] K. Keahey, P. Armstrong, J. Bresnahan, D. LaBissoniere, and P. Riteau. Infrastructure outsourcing in multi-cloud environment. In *Workshop on Multi-cloud Applications and Federated Clouds*, Sept. 2012.

[35] V. Kompella, J. Pasquale, and G. Polyzos. Multicast routing for multimedia communication. *IEEE/ACM ToN*, 1(3):286–292, Jun 1993.

[36] V. Lindberg. ZeroVM: smaller, lighter, faster. http://www.rackspace.com/blog/zerovm-smaller-lighter-faster/, Oct. 2013.

[37] C. Metz. Google open sources its secret weapon in cloud computing. http://www.wired.com/2014/06/google-kubernetes/, June 2014.

[38] Microsoft Azure. Data transfers pricing details.

http://azure.microsoft.com/en-us/pricing/details/data-transfers/, July 2014.

[39] Microsoft Azure. Virtual machines pricing details. http://azure.microsoft.com/en-us/pricing/details/virtual-machines/, July 2014.

[40] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. Sparrow: distributed, low latency scheduling. In *ACM SOSP*, Nov. 2013.

[41] F. Paraiso, P. Merle, and L. Seinturier. Managing elasticity across multiple cloud providers. In *Workshop on Multi-cloud Applications and Federated Clouds*, Apr. 2013.

[42] Rackspace. Managed VMs and bare-metal servers in the cloud. http://www.rackspace.com/cloud/servers/, July 2014.

[43] C. Raiciu, M. Ionescu, and D. Niculescu. Opening up black box networks with CloudTalk. In *USENIX Hot Topics in Cloud Computing (HotCloud)*, June 2012.

[44] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *ACM STOC*, May 1997.

[45] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: informed internet routing and transport. *Micro, IEEE*, 19(1):50–59, Jan 1999.

[46] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. In *ACM EuroSys*, Apr. 2013.

[47] Skype. How much bandwidth does Skype need? https://support.skype.com/en/faq/FA1417/how-much-bandwidth-does-skype-need, June 2014.

[48] P. Tarng, K. Chen, and P. Huang. On prophesying online gamer departure. In *ACM NetGames*, Nov. 2009.

[49] I. Vazquez-Abrams. Stackexchange superuser. http://superuser.com/questions/289978/whats-the-minimum-network-latency-for-a-1000-km-connection-using-optic-fibers, May 2011.

[50] A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues. Orchestrating the deployment of computations in the cloud with Conductor. In *USENIX NSDI*, April 2012.

[51] M. P. Wittie, V. Pejovic, L. Deek, K. C. Almeroth, and B. Y. Zhao. Exploiting locality of interest in online social networks. In *ACM CoNEXT*, Nov. 2010.

[52] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11), July 1971.