Computers & Operations Research 57 (2015) 73–82

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

The service overlay network design problem for interactive internet applications

Yuh-Rong Chen [a], Sridhar Radhakrishnan [a], Sudarshan Dhall [a], Suleyman Karabuk [b,*]

[a] School of Computer Science, The University of Oklahoma, Norman, OK, USA
[b] School of Industrial & Systems Engineering, The University of Oklahoma, Norman, OK, USA

ARTICLE INFO

Available online 24 November 2014

Keywords:
Network design
Large scale optimization
Interactive network application
Service overlay network
Communication network

ABSTRACT

Consider a private network of geographically dispersed computers with fast and high capacity connections, and an Internet application session, such as a massive multiplayer online game, with a server and a set of clients. We refer to the former as a service overlay network (SON), and assume that it could be connected to the Internet. The problem is to decide how to configure and utilize the SON in support of this application, such that the clients' speed of communication with the server is within given communication performance requirements.

We provide an Integer Programming formulation of this problem, and prove that it is $\mathcal{NP}-$Hard. In an attempt to solve the problem within strict computational time requirements of actual applications, we develop a solution framework based on partitioning and enumerating the solution space into smaller subproblems, one or more of which contains an optimal solution. In this framework, we develop and test an optimal seeking exact, and a fast polynomial time heuristic algorithm with success. The exact algorithm sets optimally solvable sizes of the subject problem, whereas the heuristic algorithm sets the size of solvable instances in a real application.

Published by Elsevier Ltd.

1. Introduction

Network applications that involve a large number of users who participate in a communication session have become more popular today with the availability of affordable high-speed Internet. Some of these applications are interactive, meaning that two-way communication takes place between users, such as distributed multimedia collaboration [13], networked audio/video conferences [11,18], multiplayer online games [2,3,23,27] (including online virtual worlds [7]), and online auctions [29]. Whereas some are non-interactive such as Internet radio, and movie streaming. This research is concerned with interactive network applications, where participation takes place over the Internet.

A multiplayer online game is arguably the most widely utilized application type thanks to the pervasive availability of game consoles with Internet capability such as Xbox or PlayStation and their corresponding online multiplayer gaming services, which have enabled a wide-range of individuals to participate in a game. It is also the largest application type with a number of participants reported to be as many as 4000 [21], although a typical session within a game is usually limited to several hundreds. For ease of exposition, throughout the rest of the manuscript we will utilize a multiplayer online game (MOG) application as context when needed.

Consider p users involved in a MOG session. In order to keep a consistent view of the common virtual world, every user (or client)[1] sends its updates such as locations or actions, in the format of network packets, to other users and receives updates from others. As illustrated in Fig. 1, there are two major network models to facilitate this interaction: centralized (or client–server), where all communications from users are routed through a server, and decentralized (or Peer-to-Peer), where users communicate directly without any central coordinator.

In the centralized model (Fig. 1a), a global state of the common virtual world is maintained by the central server S. The users send their updates to S, which in turn restores the order of these updates, computes the new global state, and sends global state updates to affected users. On the other hand, a replication of the global state is maintained by each user in the decentralized model (Fig. 1b). The users exchange their updates directly and utilize

* Corresponding author.
E-mail address: karabuk@ou.edu (S. Karabuk).

[1] To be precise, the term *client* is the computing device involved in the application and *user* is the entity that controls the client. We will use these two terms interchangeably in the paper but readers should be able to distinguish them from the context.

http://dx.doi.org/10.1016/j.cor.2014.11.003
0305-0548/Published by Elsevier Ltd.

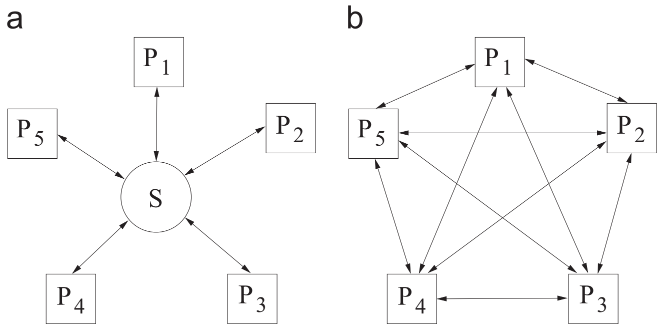

Fig. 1. Two communication models of interactive network applications. (a) Centralized and (b) decentralized.

more complex algorithms to restore the order of the events before computing the new global state. The decentralized model is known to be less scalable due to the lack of a centralized control [16]. The focus of this research is on the design of centralized networks.

During a MOG session a player performs an action using input devices. This action must take effect within a short period of time, otherwise users may stop playing and leave the game [10]. The time difference between when an action is performed and when it takes effect is called the perceived latency, most of which is attributable to the network (i.e. the Internet) latency. An update needs to travel to the central server and the corresponding global state update comes back to the users over the Internet. The perceived latency can be roughly estimated as twice of the network latency from a user to the central server. Therefore, the duration of perceived latency, which must be set to an acceptable value and be satisfied for all players, is one critical performance measure of the application [12]. Henceforth, we refer to this consideration as the delay.

Consider a MOG session with the central server *S* and two users *A* and *B*, such that the network delay from *A* to *S* is significantly shorter than that from *B* to *S*. At some point of the game session, *A* and *B* find some desirable object (e.g. some powerful weapon) about the same time, and they both attempt to grab it simultaneously. To do this, both players need to send a state update to *S*. Since the delay from *A* to *S* is shorter, *S* will receive *A*'s packet first and allow *A* to pick up the object. Later, when *S* receives the packet from *B*, it will deny *B*'s state update request because the object is no longer available. Furthermore, as long as *B*'s delay is significantly longer than that of *A*'s, *B* will be at a permanent disadvantage in the competition against *A*.

Another example can be observed in an online auction, where many users place (automated) bids for an item in a short amount of time, for example during the last few seconds of the auction. A user who receives updates of the currently maximum bid disproportionately faster than the others would have an information advantage that can be exploited to win the auction. Therefore, the magnitude of variation in delay among users is another critical performance metric, specifically in applications where competition is a major element. Henceforth, we refer to this consideration as the delay variation.

We adapt the definition of delay variation as the difference between the maximum delay and the minimum delay between all pairs of users, as was first mentioned in [25]. Approaches for reducing the delay variation, such as finding alternative paths in the computer network or packet buffering in the server, are presented in [5,25].

The maximum acceptable levels of the delay and the delay variation vary from application to application, even under different scenarios within the same application. For example, “raiding” in a massive MOG (wherein a large amount of users assemble at a location to perform fast-paced game actions) has more stringent delay requirements in comparison to actions involving one player simply roaming and exploring in the game world.

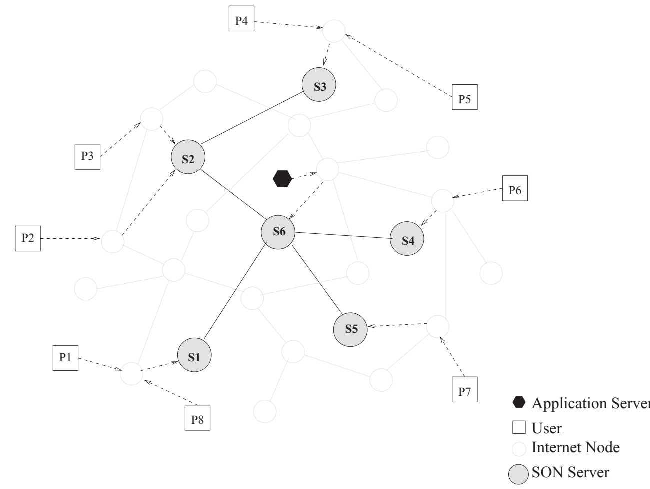

Fig. 2. Service overlay network.

In a typical application, the delay can vary greatly depending on where and when a user connects to the Internet, the routing policy of the user's Internet Service Provider, and the location of the server on the Internet. None of these factors that effect delay can be controlled or mitigated easily, which leads to the utilization of so called service overlay networks that we discuss next.

1.1. *The service overlay network approach*

It is a well established approach to utilize a private computer network of servers that are interconnected by low latency, high speed, and high capacity links in support of interactive Internet applications [1,14,26,28]. The main idea is that both the users of the network application, and the server connect to this private network via the Internet, after which all communication takes place through the fast private network. This private network is named as a service overlay network (SON).

As depicted in Fig. 2, in the proposed system, a user's delay consists of three parts: (1) the delay from the user to the SON server that the user connects to (the contact server), which takes place over the Internet, (2) the delay from the contact server to the SON server that the application server connects to (the root server), which takes place over the SON; (3) delay from the root server to the application server, which takes place on the Internet.

Therefore, bypassing the slow Internet via the SON, the performance of an interactive Internet application can be controlled and improved significantly. The first delay component can be minimized by selecting contact servers, and an assignment of users to the selected contact servers; the third delay, similarly, can be minimized by selecting a root server close to the application server. The second delay component can be affected by selecting a configuration of the SON including number and location of servers and the links in between. In this decision problem the constraints are to satisfy the delay, and the delay variation requirements, with the objective of minimizing the size of the SON, hence the total of set up and operating costs.

Another notable benefit of a SON is that the servers can be configured to conserve bandwidth via application layer multicasting [4,19,24], much more effectively than possible compared to