

# On-line Algorithm for Server Selection of Video Streaming over P2P Networks

Hao Wang

Dept. of CSIS, Kennesaw State University

hwang@kennesaw.edu

Andras Farago

Dept. of Computer Science, The University of Texas at Dallas

farago@utdallas.edu

## Abstract

We consider a video streaming application over P2P networks and introduce the P2P on-line server selection problem. This task is related to the classic K-server problem, known from the theory of online algorithms. The difference is that our problem has a dynamic cost matrix, due to network congestion which may trigger the rerouting of streaming paths. With the assumption that the number of hops of any alternative path from the server node  $p$  to the requesting node  $q$  is at most  $\delta$  times the number of hops of the shortest path from  $p$  to  $q$ , our dynamic work function algorithm works for the P2P on-line server selection problem with competitive ratio  $\delta \times (2k-1)$ .

## 1. Introduction

In this paper we consider a video streaming application over P2P networks. As we know, it is difficult to provide large scale streaming service using traditional, centralized client-server architecture, since a bottleneck occurs at the server. The network service model known as Peer-to-Peer (P2P) solves this problem, due to its distributed nature. We address the issue of optimizing the server selection for video streaming in P2P networks.

Optimization tasks are classified into online and offline problems. Optimization problems in which the input is received piece by piece and the corresponding piece of output must be produced before knowing the entire input are called online problems. In some optimization tasks, such as job scheduling, both the online and offline versions of the problem are meaningful. Many other situations, such as telephone circuit switching, investment planning etc. are intrinsically online. For these problems,

offline algorithms are not acceptable.

Competitive algorithms were introduced by Sleator and Tarjan [1] in the context of optimization problems. They sought a worst case complexity measure for on-line algorithms that have to make decisions based on current and previous parts of the input. In the on-line problem the input is given as a sequence  $\sigma = \sigma_1 \sigma_2 \sigma_3 \dots \sigma_i$  of requests. An on-line algorithm dealing with this input sequence must generate a response sequence  $\alpha = \alpha_1 \alpha_2 \alpha_3 \dots \alpha_i$ . The response must depend on only the input seen so far. The optimal off-line cost is the cost of the best output for the given input. An on-line algorithm is  $c$ -competitive if and only if

$$Cost_{on-line}(C_0, \sigma) \leq c \times Cost_{opt}(C_0, \sigma)$$

The above definition requires that the cost of the on-line algorithm,  $Cost_{on-line}(C_0, \sigma)$ , will not exceed a constant  $c$  times the cost of optimal off-line algorithm,  $Cost_{opt}(C_0, \sigma)$ , for any input sequence  $\sigma$  and any initial configuration  $C_0$ .

## 2. K-Server Problem and Work Function Algorithm

The K-server problem has been proposed by Manasse, McGeoch and Sleator [5]. An algorithm controls  $k$  mobile servers. The algorithm is presented with a sequence  $\sigma = r_1, r_2, \dots, r_n$  of requests. Using moving servers, the algorithm must serve all requests sequentially. To serve a request, a server must move to the location of the request. Moving a server, however, involves a cost that depends on the distance the server moved, this constitutes the online cost. The  $k$ -server problem revolves around the question of finding competitive online  $k$ -server algorithms for arbitrary and special metric spaces. Manasse,

McGeoch and Sleator formulated the k-server conjecture: For any metric space there is a deterministic k-competitive k-server algorithm. In other words, the cost can be made bounded by a constant factor times the optimal cost. Note that the optimal cost assumes the knowledge of the entire request sequence *in advance*.

Koutsoupias and Papadimitriou [4] have shown that there is a generic k-server algorithm, called *work function algorithm*, that is (2k-1) competitive in any metric space.

A configuration of any k-server problem is viewed as a set of k nodes in a graph G, representing the locations of the servers. The configuration is denoted by capital letters while lowercase letters are used to represent nodes. For configuration  $C_1$  and  $C_2$ ,  $C_1 + C_2$  denotes the multi-set join; similarly,  $C_1 - C_2$  denotes the multi-set exclusion. For a node  $p$  and configuration  $C$ , we abbreviate  $C + \{p\}$  by  $C + p$  and  $C - \{p\}$  by  $C - p$ . For any two configuration X and Y, we define  $D(X, Y)$ , the configuration distance, as the value of the minimum weight matching between X and Y.

For a request sequence  $\sigma = r_1, r_2 \dots r_n$ ,  $\sigma_i$  is used to denote the prefix  $r_1, r_2 \dots r_i$  of  $\sigma$ . For each initial configuration  $C_0$ , configuration  $C$  and  $i=1, 2, \dots$ , the k-server work function  $w_{\sigma_i}(C) = w_{\sigma_i}(C_0, C)$  is defined as the optimal offline cost of sequentially servicing all the requests in  $\sigma_i$ , starting from configuration  $C_0$  and ending at configuration  $C$ . Suppose  $C$  is any configuration and let  $x, y$  be any two points. The expression  $w(C - x + y) + d(y, x)$  can be interpreted as the cost of optimally servicing the request sequence thus far while ending in the configuration  $C - x + y$  and then moving to the configuration  $C$  by moving a server from  $y$  to  $x$ . This cost is no smaller than the cost  $w(C)$  of optimally servicing the same request while ending in the configuration  $C$ . That is,

$$w(C) \leq w(C - x + y + d(y, x))$$

Work functions can be computed recursively as follows. For each configuration  $C$ , the initial work function  $w_\phi(C)$  is simply the configuration distance between  $C_0$  and  $C$ . That is,  $w_\phi(C) = D(C_0, C)$ . Assume that the value of  $w_{\sigma_i}(C)$  is known for any configuration  $C$ . Given the next request  $r = r_{i+1}$  and a configuration  $C$ , the value of  $w_{\sigma_{i+1}}(C)$  is computed as follows. If  $r \in C$ , then clearly  $w_{\sigma_{i+1}}(C) = w_{\sigma_i}(C)$ . Otherwise,  $r \notin C$  and the optimal offline algorithm must first serve the request  $r$  before ending up in configuration  $C$ . The optimal offline algorithm can first process the sequence  $\sigma_i r$  ending up in some configuration  $B = C - x + r$  that contains  $r$  (differs from  $C$  by one node) and then move to configuration  $C$ . In other words,

$$\begin{aligned} w_{\sigma_{i+1}}(C) &= \min_{x \in C} \{w_{\sigma_i}(C - x + r) + d(r, x)\} \quad (1) \\ &= \min_{x \in C} \{w_{\sigma_i}(C - x + r) + d(r, x)\} \end{aligned}$$

the second equality in the Equation 1 is due to the fact the  $r \in C - x + r$ . If we use the notation

$$\begin{aligned} w(C) &= w_{\sigma_i}(C) \\ w'(C) &= w_{\sigma_{i+1}}(C) \end{aligned}$$

Then we obtain the basic recurrence for the work function:

$$\begin{aligned} w_\phi(C) &= D(C_0, C) \\ w'(C) &= \min_{x \in C} \{w(C - x + r) + d(r, x)\} \quad (2) \end{aligned}$$

Equation 2 provides the means for computing the work functions. This can be done efficiently using dynamic programming.

The classic on-line k-server work function algorithm works as follows:

Let  $\sigma_i$  be the request sequence thus far and let  $C$  be the configuration of work function algorithm after servicing  $\sigma_i$ . Then, given the next request  $r = r_{i+1}$ , the

work function algorithm serves  $r$  with a server  $s \in C$  satisfying

$$s = \arg \min_{x \in C} \{w(C - x + r) + d(x, r)\}$$

with ties broken arbitrarily.

For any  $k$  and any metric space, the work function algorithm is  $(2k - 1)$  competitive, that is

$$Cost_{on-line}(C_0, \sigma) \leq (2k - 1) \times Cost_{opt}(C_0, \sigma)$$

### 3. P2P Video Streaming Service Model and Problem Formation

Suppose an undirected graph  $G$  represents a P2P network, supporting a video streaming application. For P2P video streaming applications, a node could be considered as a server node if it has the desired video content. Nodes that do not have the desired video content are considered non-server nodes. A node participates in a streaming video process if it is regarded as an intermediate hop along the video streaming path. In the P2P video streaming described above, the on-line server selection problem is defined as follows: Given a streaming request sequence  $\sigma = r_1, r_2, \dots, r_n$  at nodes in network  $G$ , is the work function algorithm appropriate to solve the server selection task? If not, is there any other competitive on-line server selection algorithm?

In order to define the on-line server selection problem properly, we assume that

(1) Each edge in the network  $G$  has unit weight and the network topology remains unchanged through the entire request sequence. Initially, the server nodes are located randomly among clients in the network  $G$ . Each server node has a copy of the video file.

(2) Streaming requests are sequential, which means at any time at most one request can be issued and served. The next request will not be produced until one of the server nodes starts to serve the current request.

(3) Only the non-server nodes can request video streaming service (We do not consider the requests

generated at server nodes, since the request occurring at server nodes can be served locally). When a new request at client  $p$  has been served,  $p$  becomes a new server node, since it already has the video file.

(4) Due to the limited computing capability, a node which contains the desired video content can provide streaming service to at most one request. Therefore a server node will not be available for future requests once it starts to serve a previous streaming request.

(5) It is always true that, from assumption (3) and (4), the number of available server nodes in the network  $G$  remains constant.

(6) A node becomes congested if it does not have enough bandwidth available to support further video streaming applications.

(7) The cost of serving a new request at node  $p$  by a server node  $q$  is the weight (the number of hops) of streaming path from  $p$  to  $q$ .

The classic  $K$ -server problem has fixed cost matrix, so the cost of serving a request at node  $p$  by a node  $q$  remains unchanged. However, in our P2P on-line server selection problem, due to network congestion, the streaming video might be detoured to an alternative path. Therefore, streaming path may not be the shortest path between  $p$  and  $q$  and the cost is subject to change dynamically, which makes the P2P on-line server selection problem different from the  $K$ -server problem.

### 4. Solution to the proposed on-line server selection Problem

As an example, consider a 5-node weighted undirected graph  $G$  with node set  $\{a, b, c, d, e\}$ . Assume that three servers are initially located on nodes  $a, b$  and  $c$ . In the dynamic programming table shown in Figure 1, we specify values of work functions corresponding to all 3-node configurations and all prefixes of the request sequence  $e, d, a, b, c, a, b, a, c, e$ .

Case 1: No Congestion

Assume that no node becomes congested, therefore the cost of serving a request at node  $p$  by a node  $q$

remains constant and equals to the number of hops of the shortest path from  $q$  to  $p$ , as we mentioned in assumption (7). Then case 1 is the classic  $k$ -server problem, which can be solved by the work function algorithm with competitive ratio  $(2k - 1)$ .

Servers Configuration										request
abc	abd	abe	acd	ace	ade	bcd	bce	bde	cde	
										$\Phi$
										e
										d
										a
										b
										c
										a
										b
										a
										c
										e

Figure 1. dynamic programming table of robust work function algorithm

We assume that the on-line algorithm works in a distributed manner in the proposed P2P video streaming service model. Every node in the network  $G$  maintains global information, including the network topology, server distribution, request sequence, node congestion, etc. The on-line computation of server selection is performed by the requesting node. When a server node is chosen, the requesting node generates an update message. The update message contains the server node, requesting node and streaming path information associated with a particular service request  $r$ . Then the requesting node broadcasts the newly constructed update message. Upon receiving the update message, each node updates its global information so that all nodes are synchronized. Let us now consider the case of packet loss. If the update message is corrupted or lost, some nodes that fail to receive the original update message will lose synchronization, resulting in removal of the service request  $r$  from the original request sequence. Therefore, the on-line algorithm

deals with a subsequence of the original request sequence instead of the original request sequence. When the classic work function algorithm is applied to the subsequence of the request sequence, the computation based on the dynamic programming table will be changed correspondingly. To be more accurate, in the dynamic programming table a request that corresponds to the lost update message will be eliminated from the original table. Figure 1 shows this elimination.

Entries in the dynamic programming table are defined as  $w_{\sigma_i}(C)$ , where  $C$  is the server configuration and  $\sigma_i$  is the prefix,  $r_1, r_2, \dots, r_i$  of  $\sigma$ . The dynamic programming table is non-decreasing, that is,  $w_{\sigma_i}(C)$  is no less than  $w_{\sigma_{i-1}}(C)$ . Computing the row of  $r_{i+1}$  in the dynamic programming table is based on  $w_{\sigma_{i+1}}(C) = \min_{x \in C} \{w_{\sigma_i}(C - x + r) + d(r, x)\}$ .

If the update message associated with request  $r_i$  is corrupted or lost, the row  $i$  need to be eliminated from the dynamic programming table. Computation of row  $i+1$  will be based on

$$w'_{\sigma_{i+1}}(C) = \min_{x \in C} \{w_{\sigma_{i-1}}(C - x + r) + d(r, x)\}$$

For example, in Figure 1, if request  $b$  is corrupted or lost, the corresponding row in the table will be eliminated. Since  $w_{\sigma_i}(C)$  is no less than  $w_{\sigma_{i-1}}(C)$ ,  $w'_{\sigma_{i+1}}(C) \leq w_{\sigma_{i+1}}(C)$ . Therefore after elimination of row  $i$  from the original dynamic programming table  $T$ , all rows after row  $i$  will be re-computed and the new dynamic programming table  $T'$  has the following property:

- (1) from the first row to row  $i$ , each element of  $T'$  is identical to that of  $T$
- (2) for all rows after the row  $(i+1)$ , each element of  $T'$  is no less than that of  $T$

Based on the above property, it is straightforward that

$$Cost_{on-line}(C_0, \sigma') \leq Cost_{on-line}(C_0, \sigma) \leq c \cdot Cost_{opt}(C_0, \sigma)$$

Basically, the inequality above states that for the request sequence  $\sigma'$  which is a subsequence of the original request sequence  $\sigma$ , the on-line cost of  $\sigma'$  is bounded by the optimal cost of  $\sigma$  multiplied by the

constant  $c$ . The work function algorithm working on a subsequence of request sequence is still  $(2k-1)$  competitive. Therefore the work function algorithm is robust to update message corruption or loss.

### Case 2: Congestion

Assume that each streaming request can be served by at least one server node no matter how deteriorated the network congestion is. It is a reasonable assumption when network has sufficient resource and there is a fair number of streaming requests.

Note that the streaming path between requesting node  $p$  and the server node  $q$  may not be the shortest path from  $p$  to  $q$  due to network congestion. Obviously, the cost of servicing request will be changed, which introduces additional complexity to build up the dynamic programming table of the work function algorithm. In classic K-Server problem, for each request at node  $r$ , any server node  $x$  in the configuration  $C$  will be tested and  $d(r, x)$  remains unchanged, as referred in Equation 2. In this case, due to network congestion  $d(r, x)$  might be changed. Therefore,  $d(r, x)$  is replaced by  $d'(r, x)$  and Equation 3 is obtained. Similarly, computation can be done using dynamic programming table.

$$w_\phi(C) = D(C_0, C)$$

$$w'(C) = \min_{x \in C} \{w(C - x + r) + d'(r, x)\} \quad (3)$$

Note that  $d'(r, x)$  is a family (that may contain multiple copies of same value) of values each of which is the cost associated with a particular streaming path between requesting node  $r$  and server node  $x$ . To differentiate elements in the family, each element is denoted by a vector (value, path). The “value” is the cost and the “path” is the associated path. Generally, since there is an exponential number of streaming paths between a pair of nodes, direct computation based on Equation 3 would have exponential time complexity, which is not applicable.

**Lemma:** Suppose  $P_a$  is an alternative path (other than the shortest path  $P_s$ ) between nodes  $p$  to  $q$ . The number of hops of  $P_a$  is equal to the summation of the number of hops of a series of paths. These paths

are the shortest path of nodes  $p$  and  $s_1$ , of nodes  $s_1$  and  $s_2$ ...of nodes  $s_n$  and  $q$  respectively.  $s_1, s_2, \dots, s_n$  are intermediate nodes of  $P_a$ , such that sub-path  $p - s_1$  along  $P_a$  has the same number of hops as the shortest path from  $p$  to  $q$  makes,  $s_1 - s_2$  has the same number of hops as the shortest path from  $s_1$  to  $q$  makes, and so on.

Proof:

In Figure 2,  $p$  is the responding server node and  $q$  is the requesting node.  $P_s, P_a$  are respectively the shortest path and an alternative path between  $p$  and  $q$ .

The number of hops of  $P_a$  is larger than that of  $P_s$ . An intermediate node  $s_1$  along  $P_a$  can be found such that the sub-path  $p - s_1$  has the same number of hops as that of  $P_s$  (if the cost of  $P_a$  has the same number of hops as that of  $P_s$ , then node  $s_1$  is node  $q$ ). Next, if the sub-path from  $s_1$  to  $q$  along  $P_a$  is not the shortest path from  $s_1$  to  $q$ , repeat the previous process to find another intermediate node  $s_2$  between nodes  $s_1$  and  $q$ . The iterative process will terminate when the sub-path from  $s_n$  to  $q$  is the shortest path of nodes  $s_n$  and  $q$ . Since  $P_a$  has a finite number of hops, therefore, the number of iterations is finite. Consequently,  $P_a$  can be regarded as the concatenation of sub-paths  $p - s_1, s_1 - s_2 \dots s_n - q$ . Note that sub-paths  $p - s_1, s_1 - s_2 \dots s_n - q$  are the shortest paths between the corresponding pairs of nodes. This proves the Lemma.

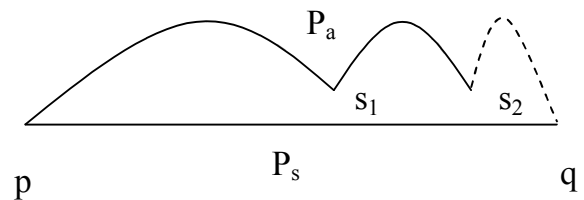


Figure.2 Concatenation of Alternative Path  $P_a$

When the shortest path  $P_s$  between the server node  $p$  and the requesting node  $q$  is congested, the video

streaming traffic will be detoured to an alternative path  $P_a$ . To avoid computing exponential number of alternative paths,  $P_a$  is defined as the shortest path  $P'_s$  from p to q, after removal of congested nodes in the network graph G.

Any server can be chosen to service the request r. Therefore, there are k possible streaming paths and each corresponds to one particular server x. The computation of  $d'(r, x)$  can be done in polynomial time. Therefore the 2-D dynamic programming table of the classic K-Server work function algorithm becomes a 3-D dynamic programming cube, which is shown in Figure 3. The third dimension of the dynamic programming table is server nodes.

Note that each of k streaming paths could be either an alternative path  $P'_s$  or the shortest path  $P_s$ . For an alternative path  $P'_s$ , request q will be replaced by a series of requests at nodes  $s_1, s_2, \dots, q$ . Nodes  $s_1, s_2, \dots$  are intermediate nodes along  $P'_s$  and can be computed based on the Lemma. Therefore  $P'_s$  is decomposed into a series of shortest paths. In the dynamic programming table, the row with respect to request q will be expanded into a series of rows corresponding to  $s_1, s_2, \dots, q$ , which is shown in Figure 4(b). We call it dynamic programming table expansion. In the 3-D dynamic programming cube, element  $(x, y, z)$  represents the associated cost after servicing request x by server node z with server configuration y. An element  $(x, y, z)$  might be decomposed into a series of elements  $(x_1, y, z)$   $(x_2, y, z)$  ...if request x is replaced by a series of requests  $x_1, x_2, \dots$ .

For each request r, each server candidate will be examined and a server x with the minimum number of hops will be chosen on-line. The computation of 2-D dynamic programming table corresponding to current request r and server x is performed. Note that 2-D dynamic programming table might be an expanded dynamic programming table when the streaming path is not the shortest path between r and x.

$P'_s$  is the shortest path when congested nodes are removed from network G. It is well known that a video streaming application has end-to-end delay constraint. We assume that the end-to-end delay is proportional to the number of hops of the streaming path. If the number of hops of any alternative path  $P'_s$  is bounded by the number of hops of the corresponding shortest path  $P_s$  multiplied by a bound factor  $\delta$  (due to delay constraint), then we have

$$\text{MAX} \left\{ \frac{|P'_s|}{|P_s|} \right\} \leq \delta$$

where

$|P'_s|$  is the number of hops of any alternative path  $P'_s$  from p to q  
 $|P_s|$  is the number of hops of the shortest path  $P_s$  from p to q

**Theorem:** The proposed dynamic on-line k-server work function algorithm works for the congested network with competitive ratio  $\delta \times (2k - 1)$ , where  $\delta$  is the bound factor for any alternative path  $P'_s$ .

Proof: The dynamic programming table for the classic K-Server problem is shown in Figure 4 (a). The on-line and off-line costs associated with the original request sequence  $r_1, r_2, r_3, \dots$  are  $C_{on-line}$  and  $C_{off-line}$  respectively. After dynamic programming table expansion, the expanded dynamic programming table is shown in Figure 4 (b). The on-line and off-line costs associated with the decomposed request sequence  $r_1, r_2, r_3, r', r_4, \dots$  are  $D_{on-line}$  and  $D_{off-line}$  respectively. Note that request  $r_4$  is replaced by requests  $r'$  and  $r_4$ . From the working function algorithm, we know

$$C_{on-line} \leq (2k - 1) \times C_{off-line}$$

$$D_{on-line} \leq (2k - 1) \times D_{off-line}$$

## References

- [1] Fiat, A, Rabani, Y, Ravid, Y, Proceedings 31<sup>st</sup> Annual Symposium on Foundations of Computer Science, Oct, 1990, Vol 2, pp. 454-463
- [2] Fiat, A, Ricklin, M, Proceeding of the 2<sup>nd</sup> Israel Symposium on Theory and Computing Systems, June 1993, pp. 294-303
- [3] Koutoupias, E, Papadimitriou.C.H, Proceedings of the 35<sup>th</sup> Annual Symposium on Foundations of Computer Science, Nov 1994, pp. 394-400
- [4] Koutoupias, E, Proceedings of 40<sup>th</sup> Annual Symposium on Foundations of Computer Science, Oct, 1999, pp. 444-449
- [5] Borodin. A, El-Yaniv, R, *Online Computation and Competitive Analysis*, Cambridge University Press, 1998
- [6] D.D.Sleator, R.R.Tarjan, "Amortized efficiency of list update and paging rules", Comm, ACM 28, No.2, 1985, pp. 202-208.

$C_{off-line}$  is the optimal off-line cost of classic work function algorithm when no congestion is considered. The optimal off-line algorithm selects a series of servers, named as  $S_{off-line}$ , to service request sequence.  $D'_{off-line}$  is defined as the cost when choosing the same series of servers  $S_{off-line}$  to service the same request sequence in a congested network.  $D'_{off-line}$  is not off-line optimal. From our assumption that  $|P'_s|$  is bounded by the multiplication of  $|P_s|$  and  $\delta$ , we have the following inequality:

$$D'_{off-line} \leq \delta \times C_{off-line}$$

$D_{off-line}$  is the optimal off-line cost when congestion is considered. Obviously,  $D_{off-line} \leq D'_{off-line}$  holds. Then

$$D_{off-line} \leq \delta \times C_{off-line}$$

Hence,

$$D_{on-line} \leq (2k-1) \times \delta \times C_{off-line}$$

Then the theorem follows.

## Conclusion

The classic K-Server work function algorithm works only with fixed cost matrix. However, real network applications, such as video streaming application in P2P network, suffer from packet loss and network congestion so that the cost matrix cannot remain fixed. The proposed dynamic on-line K-Server work function algorithm is robust and competitive with competitive ratio  $\delta \times (2k-1)$ , given that the number of hops of all alternative path  $P'_s$  is bounded by the number of hops of the original shortest path  $P_s$  multiplied by a bound factor  $\delta$ .

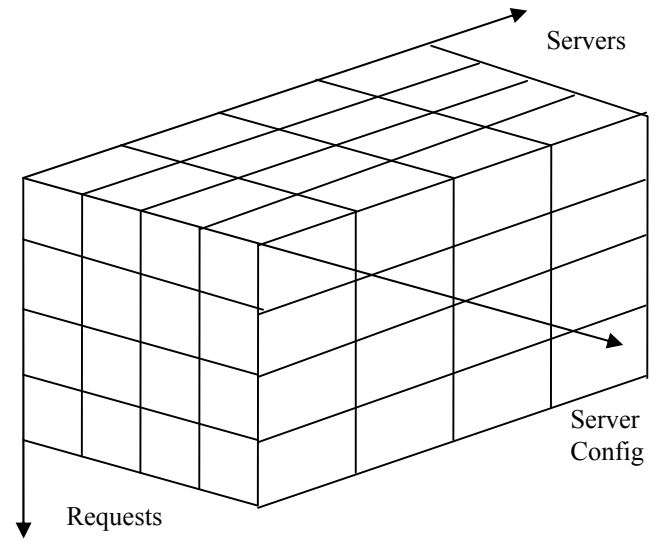


Figure 3. 3-D dynamic programming cube

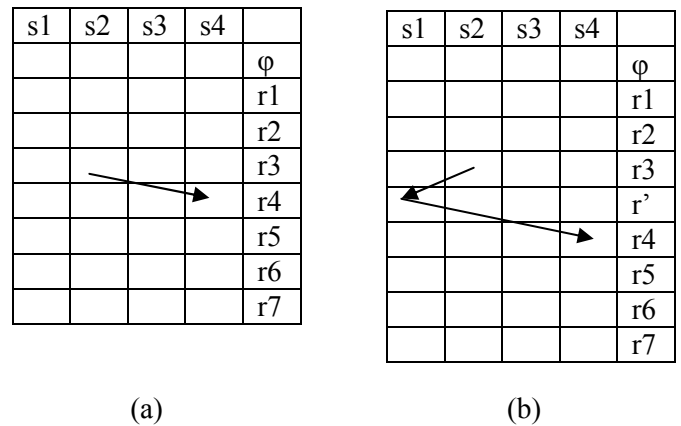


Figure 4. 2-D dynamic programming table expansion