Improving the QOS in Video Streaming Multicast

Sujatha M.
Assistant Professor, St. Joseph Engineering College, Vamanjoor,Mangalore,
Karnataka, India-575028.
Email: sujatha_msk@yahoo.co.in

Abstract

In a streaming video multicast environment a large number of users often request various similar processing on the same stream. Therefore service sharing is feasible, with a large potential of savings in processing cost.

The Service invocation order may play a significant role on service sharing in a streaming video multicast application. The problem is how to determine Service Invocation Order for multiple service composition requests with the aim of maximizing service sharing. This paper proposes BFS algorithm to maximize the service sharing in a streaming video multicast environment in order to reduce the processing cost and to increase the scalability. The service invocation orders for multiple service composition requests will be determined with the aim of maximizing the service sharing. The BFS algorithm merges the services level by level without changing the service invocation order.

Keywords: Service invocation order, service composition graph, Greedy algorithm

1. Introduction

With the rapid growth of the Internet and multimedia systems in distributed environments, it is easier for digital data owners to transfer multimedia documents across the network. Therefore, there is an increase in concern over streaming of digital contents.

Streaming is the process of transferring data via a channel to its destination with real time characteristics, where it is decoded and consumed via a user/device in real time, i.e.,as the data is being delivered on the fly [1], [2]. It differs from non-streaming process because it does not require the entire data to be fully downloaded before it can be seen or used.

Streaming is broadly classified into two types[3].

- Live or on-demand: Live webcasts require some extra equipment. We will need an on-site computer that can compress, encode and stream the video feed in real time or a satellite uplink to a company that can do it for us.
- Unicast or multicast: In a unicast stream, each person watching gets his own stream of data. In a multicast stream, one stream of data travels to a router, which copies the stream and sends it to multiple viewers.

In a general context the sharing of intermediate service results among different processes is seldom feasible because parameters are often different and there may be transactional and side effects. However, in a streaming video multicast environment a large number of users often request various similar processing on the same stream. Therefore service sharing is feasible, with a large potential of savings in processing cost[2]. In this paper, the service invocation orders for multiple service composition requests will be determined with the aim of maximizing the service sharing. In this paper we develop an algorithm to merge the services level by level without changing the service invocation order.

This paper aims to maximize the service sharing in a streaming video multicast environment in order to reduce the processing cost, to increase bandwidth and scalability. The Service invocation order may play a significant role on service sharing in a streaming video multicast application. The problem is how to determine SIO (Service Invocation Order) for multiple service composition requests with the aim of maximizing service sharing.

Each service composition request is associated with a *service composition graph*, which specifies a set of required services.
Eg; Consider 3 services namely s1, s2, s3, s1 has to be performed first, followed by s2 and s3. The service composition graph for such a request is {s1->s2->s3}

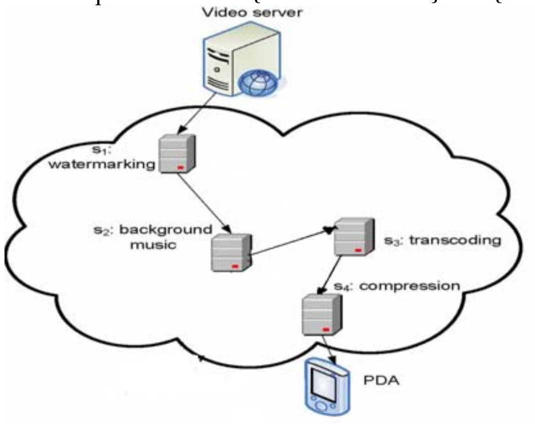
Service composition has various performance metrics[4][5]:

- Load balance

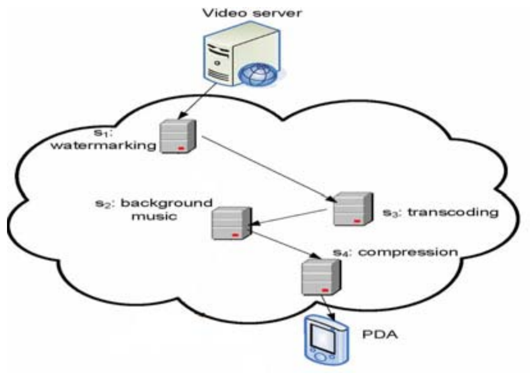
If too many people try to access a file at the same time, the server can delay the start of some streams until others have finished.

- End-to-end delay
  Unicast requires more end to end delay.
- Bandwidth
  Unicast streams require more processing power and bandwidth.
- Resource
  With multicast resources can be shared.

Services required by the end user may have alternative service invocation orders, thus multiple service composition graphs may exist for one request.
Eg; Consider that there are 4 services, namely s1,s2,s3 and s4. s1 has to be performed first and s4 at the end. The service composition graph for such a request is either {s1->s2->s3->s4} or {s1->s3->s2->s4}



{s1->s2->s3->s4}

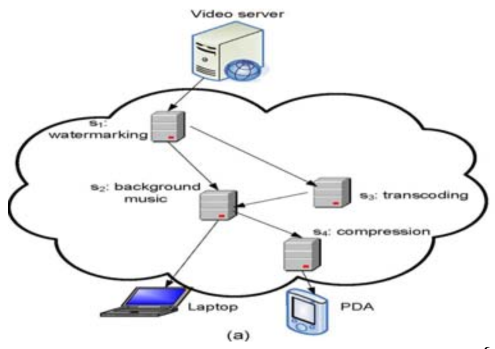


{s1->s3->s2->s4}

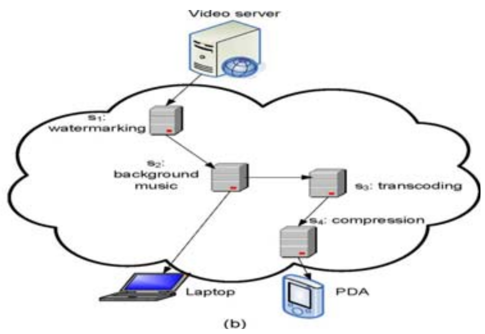
Fig. 1. Example for multiple service composition graphs.

In a streaming video multicast application, the end users may have heterogeneous end devices and require various QoS, thus the streaming video contents delivered to the end users are the results of different service composition requests.

Consider user1(Laptop) wants to watch a video(service,s0). Streaming involves set of services like watermarking(s1), background music(s2).user2(PDA) wants to watch the same video(s0). Streaming involves watermarking(s1), transcoding(s3), background music(s2), compression(s4). The intermediate services can be invoked in any order. Hence we will get different service composition graph. If the service invocation order for each user is independently determined, the service invocation order for PDA user may be {s1->s3->s2->s4} as shown in Fig 2(a). As a result only s1 is shared. If the service composition orders for two users are considered coordinately, the two users can share both s1 and s2 as shown in Fig 2(b). This example shows that service invocation order plays a significant role on service sharing in a streaming video multicast application.



{s0->s1->s2} {s0->s1->s3->s2->s4}



{s0->s1->s2->s3->s4}

Fig. 2. Example of streaming video multicast. (a) two users share a single service s1 only and (b) two users share two services s1 and s2.

Cost savings among different service composition requests can be achieved through the following three steps.

Step 1: Coordinately determine the service composition graph for each request by controlling the service invocation order for each required service in a request such that the maximum number of shared services among multiple requests is achieved.

Step 2: Merge the given service composition graphs such that each shared service will only be invoked once.

Step 3: Assign each required service in the merged service composition graph to its most appropriate service provider.

2. Related Work

The literature work on service sharing assumes that the service composition graph for each request is independently generated. Some work focuses on finding partial overlaps among simultaneous requests and some other work focuses on assigning each required service in the merged service composition graph to the most appropriate service provider.

The service composition graphs, aims to save the service and bandwidth cost. [1] proposes optimal algorithm for the base case of two requests. Then two heuristic algorithms, namely global greedy algorithm and local greedy algorithm using the optimal algorithm for the base case as the building block.

3. Problem Description

3.1. Service Sharing Conditions

(1) If the streaming video contents of the service composition requests are different, such requests can not share any service.

(2) If two service composition requests share a service such two service composition requests must share all preceding services.

(3) If the outputs of the (k-1)-th services of two requests are the same, it means that either the service invocation order of the first k services for such two requests will be fine.

3.2. Problem Description

Suppose S={s0,s1,s2,….,sm} are available streaming video processing services, where the cost to invoke service s_i is c_i. Suppose that there are n service composition requests, U={u1,u2,….,un}, which require the same **streaming** video content. Each service composition request u_i requires a set of service Si≤S. The service dependency among the services in Si can be described by a directed acyclic graph, namely Di, with a set of nodes and directed arcs where each node is associated with a service in Si and a directed arc defines the precedence constraint between two services. In this paper, we assume that the service composition requests are successive service composition.

Consider two service composition requests u_i and u_j where Li={s_{i1} s_{i2} ….-->s_{in}} is service invocation order for Di and Lj={s_{j1} s_{j2} ….-->s_{jn}} is service invocation order for Dj. If there is a k such that $s_{it}=s_{jt}$ for t=1,2,….k, then we say that u_i and u_j share the first k services. It is obvious that a larger k leads to higher potential of service sharing.

The above observation can be generalized to multiple service composition requests. Given Si and Di for i=1,2,….,n, let o={L_1(o),L_2(o),….,L_n(o)} be a set of service invocation orders where L_i(o) is the service invocation order for u_i. Then a service invocation order tree with n leaf nodes can be constructed where each node is associated with a service, and the path from the root to a leaf node represents a service invocation order for a request. Assuming that each service s_i, involves a cost c_i, then the sum of the costs for all nodes in the tree represents the cost for the service composition requests under consideration, which is to be minimized.

4. Algorithms

4.1. Existing Algorithms to get SIO with maximal service sharing:

- SIO with two requests
- Global greedy algorithm
- Local greedy algorithm
- Lower bound algorithm

Using the above algorithm it is difficult to construct maximal common DAG. The simulation results show that relative error increases with service sharing.

4.2. BFS method to merge the services:

The maximum service sharing can be achieved among the service composition requests by using BFS method to construct DAG. In this method adjacent DAGs will be compared level by level.

Algorithm: Let R be the matrix containing the requests. The requests includes service invocation order followed by end of service request and user id.

```
 S<-Matrix containing rows of S sorted in ascending order
 T<-S;
MAX<-999
 Shared_serv<-0;
 For j←1 to Max_Serv_Required do
   Temp=T(1,j);
   For i←2 to Num_Requests
      If T(i,j)=Temp and
        Temp!=-1
        Temp<-T(i,j);
        T(i,j)<-MAX;
      else
         Temp<--T(i,j);
      End
   End
End
U<-Matrix containing rows of T sorted in ascending order of user id.
```

4.3. Comparision between proposed algorithm and Greedy algorithm

The Fig 3. shows the comparision between proposed BFS algorithm and the existing greedy algorithms used to merge the service requests of video streaming.

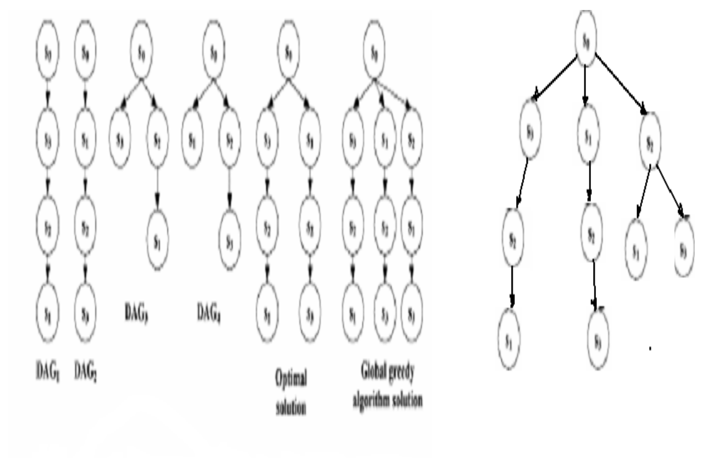


Fig. 3: Example showing optimal solution, global greedy algorithm solution and BFS algorithm solution.
Service saving in the optimal solution=4*4-3-7=6 services, global greedy algorithm=4*4-3-10=3 services**,** using BFS 4*4-3-10=3 Services.

The greedy algorithm uses topological sorting to obtain the service invocation order. The Fig.3. shows that using greedy algorithms the service invocation order of the request may alter. The service saving using BFS is equal to the global greedy algorithm. But if we merge the shared services which are at the same level the service invocation order of the requests remains unaltered.

5. Experimental results

The simulation result has been taken based on rate of service saving versus total number of services shared. The result shows that the rate of service saving increases with the number of services shared. In the simulation, there are n requests where n varies from 20 to 50. There are m services in total where m varies from 10 to 30. The cost of each service is set to 1. The number of services that each request requires is in the range M/2 to M. The performance of the BFS algorithm is compared with the existing algorithms as shown in Fig. 4.

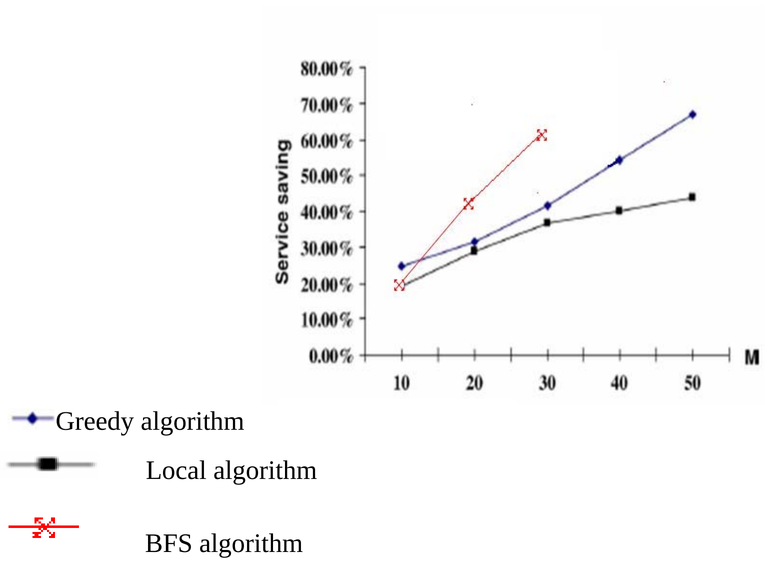


Fig. 4. Service sharing of the BFS algorithm compared with the greedy algorithms and naïve algorithm.

If there are n requests with maximum m services the time complexity of service sharing using bfs method is O(mn). Using global greedy algorithm the time complexity is O(n^2E max(D)) and for local greedy algorithm time complexity is equal to O(nEmax(D)).

The above time complexities clearly indicates that as the number of requests increases the time required for computation using BFS algorithm significantly less when compared with global greedy algorithm.

6. Conclusion

In this paper service invocation orders in a streaming video multicast with heterogeneous service composition requests will be determined such that the total cost of invoked services will be minimized. It also aims in reducing the bandwidth, hence to improve the scalability.

Acknowledgments The author wish to thank the Department of Computer Science of St. Joseph Engineering College, Vamanjoor, Mangalore, India. For their encouragement and guidance towards the completion of the work and also the management of St.Joseph Engineering College, Mangalore, India for their valuable support.

References

[1] http://www.bitecomm.co.uk/Prognet_pages/corp_backgrounder.html
[2] http://www.netlab.ohiostate.edu/~jain/cis788-97/ip_multimedia / index.htm
[3] ”A Primer on Video Streaming” http://opi.mt.gov/Streamer/index.html
[4] "A Streaming Media Primer“ http://www.adobe.com/ products/aftereffects/pdfs/AdobeStr.pdf
[5] “Digital video for the next Millennium” www.video.net/resources/whitepapers/video/7.shtml

Authors Profile

Ms.Sujatha M. is currently working as an Assistant Professor in the Department of Computer Science and Engineering, St. Joseph Engineering College Mangalore, India. She obtained her Bachelor’s degree in Computer Science and Engineering from St. Joseph Engineering College Mangalore, India.She obtained her Masters degree in Computer Science and Engineering from NMAMIT, Nitte, Karnataka, India. Her research interests are in the area of networks and image processing.