

# Resource Optimization for Web Service Composition

Xia Gao, Ravi Jain, Zufikar Ramzan, Ulas Kozat

**Abstract**—Service composition recently emerged as a cost-effective way to quickly create new services within a network. Some research has been done to support user perceived end-to-end QoS for service composition. However, not much work has been done to improve a *network operator's* performance when deploying composite services. In this paper we develop a service composition architecture that optimizes the aggregate bandwidth utilization within a operator's network; this metric is what operators care about most. A general service composition graph is proposed to model the loosely coupled interaction among service components as well as the estimated traffic that flows among them. Then an optimization problem is formalized and proved to be NP-hard, even to *approximate*. Next, two polynomial-time heuristic algorithms are developed together with several local search algorithms that further improve the performance of these two algorithms. Our simulations demonstrate the effectiveness of both approximation algorithms and show that they are suitable for service graphs with varying topologies.

## I. INTRODUCTION

Service composition is a key enabling technology for fast and flexible service deployment. It can be iterated to progressively integrate service components of different granularity and manage the complexity of service development. Web service technologies such as WSDL [1] provide tools to unify the semantics and syntax of service definition and access. They allow service composition to be carried out at the application level and seamlessly integrated into enterprise business logic.

There has been recent interest in the service composition field to guarantee Quality of Service (QoS) when distributed service components are integrated to provide one composite service. As discussed later, existing solutions only consider QoS requirements of interest to *users* of each service; the classic example of such a QoS metric is the delay experienced by the user. To derive the end-to-end QoS bound, orchestration models used by existing solutions in [2], [3] are the simplistic chain (service path) model and its combination (multicast tree). Their goal is to determine the appropriate application servers that form the given service chain and whose combined performance also meets the overall QoS requirements of each individual user. Furthermore, because these solutions assume that interactions between service components conforms to well-defined logic, they are mainly suitable for low-level service composition where behavior of each component is well defined. Consequently, they cannot be used for application-level service composition where service components are loosely coupled.

Little work has been done so far to improve a *network operator's* performance when deploying composite services. One of the QoS metrics network operators care about most is the total aggregate bandwidth utilization within its network. This metric greatly influences the efficiency with which a network operator manages its network and the number of services that can be deployed simultaneously with a given network capacity. Furthermore, according to the current trend of

service composition exhibited by BPEL [4] and similar efforts, the orchestration model to compose service components will be fairly complex. A general graph, probably cyclic in many scenarios, is required to describe complex service composition logic among service components. This can not be handled by existing solutions [2], [3], [5], [6], [7].

To this end, this paper proposes a framework that uses a generic service graph to capture loosely coupled complex service logic and optimizes the overall bandwidth usage within an operator's network to support a large number of service users.

The paper is organized as follows. Section II discusses the related work on service composition and its QoS support, which puts our work in perspective. Section III presents our proposed system of QoS support for the generic service composition model based on web service technologies. Section IV gives the formal definition of the "service mapping" problem which maps a given arbitrary service graph to a physical network and minimizes the aggregate network bandwidth usage. Section V proves the "service mapping" problem is NP-hard to even approximate and Section VI correspondingly presents two heuristic algorithms. Section VII then presents the simulation results of the proposed algorithms with different setup. Finally, section VIII gives some discussions and section IX concludes the paper with our future works.

## II. RELATED WORK

There is much work related to service overlay network, service composition and QoS support. For brevity, we only cover work that is most relevant to our research.

### A. Existing Service Composition Related Technologies

Recently, Business Process Execution Language (BPEL) [4] has been standardized; it can define the external behavior of a service as well as its internal implementation. BPEL assumes that the interfaces of the interacting web services are defined in terms of WSDL port types and interact by exchanging WSDL messages. BPEL also defines five types of structure activities, namely, *sequence*, *switch*, *pick*, *while* and *flow*. With these five orchestration activities, BPEL can define very complex web service composition logic.

Recently, there has been work on how to manage QoS provisioning for composed services [2], [3], [5], [6], [7]. Gu [2] and Xu [5] use an application-level virtual link orchestration model that organizes service components into a chain architecture; i.e., a service path. As shown in Fig. 1, composite service logic is implemented by composing the basic service components between a service entrance portal and service exit portal. Each service component has a *service level agreement* (SLA) specifying availability and response time. Each link between adjacent service components is also characterized by its availability and transmission delay. The final goal of the work is to compose a service path among

all service components that can best avoid QoS violations or mitigate such a violation should one occur.

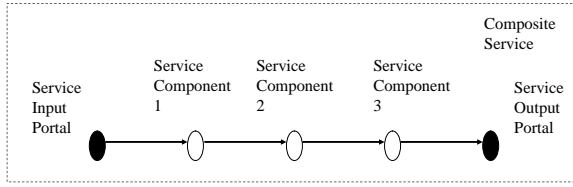


Fig. 1. Service path model of service composition.

Jin et al. [3] extend the single service path orchestration model by providing service composition functionalities to multiple users with distributed location and heterogeneous service requirements. Hence, a one-to-many multicast tree orchestration model is used. Compared with the scheme of building multiple one-to-one unicast service paths between the source and each destination node, the one-to-many multicast orchestration model can save both computing power at intermediate service nodes and bandwidth of intermediate links by combining common segments of unicast service paths. The final goal of the work is to build an optimal multicast service tree that meets bandwidth requirements and minimizes the aggregate delay and/or the total number of hops.

Jin et al. [6] also proposed an distributed and hierarchical service composition architecture. In this work, the service overlay network is first organized into a cluster architecture and then the service path finding algorithm is applied hierarchically in a divide-and-conquer fashion. The main focus of this work is to identify the uniqueness of hierarchical service routing compared with hierarchical network routing.

Mea et al. [7] extended the service orchestration model from a single service chain to a generic directed acyclic graph. Then *sFlow*, a distributed service routing algorithm, was developed to choose appropriate servers that satisfy the service logic requirement and optimize the QoS metrics.

### B. Limitations of Existing Solutions

To motivate our work, we now describe limitations of existing solutions.

- Existing work on QoS support in service composition aims to guarantee QoS metrics that each service *user* will experience (e.g., bandwidth and delay). They do not account for QoS metrics that a *network operator* cares about (e.g., aggregate bandwidth). Total aggregate bandwidth usage within a network influences the efficiency with which a network operator manages its network and the number of services that can be deployed simultaneously with a given network capacity.
- To derive end-to-end QoS bounds, existing solutions [2], [3], [5], [6], except [7], use simple orchestration models such as single chains (service path) and combinations of chains (multicast tree). According to the current trend of service composition exhibited by BPEL and similar efforts, the orchestration model to compose service components will be much more complex than a chain architecture. A general graph, probably cyclic in many scenarios, is required to describe complex service composition logic among service components. This cannot be handled by existing solutions.
- Existing work assumes a closely coupled relation between adjacent service components in the sense that the behavior at each service node and interconnecting

link is deterministic. Such an assumption is suitable for services, such as streaming, which have well-defined functional units that can be deployed at each node along the service path. But it is too restrictive for loosely coupled services such as web services.

## III. SERVICE COMPOSITION MODEL AND SYSTEM ARCHITECTURE

In previous section we have discussed the major limitations of existing service composition models. In this section we at first present a new model that is more suitable for web-technology based service composition. We then propose a system architecture that enables a network operator to improve its network efficiency when composing distributed services.

### A. Generic Service Composition Model

Figure 2 shows the service composition graph that a network operator uses to support the business of a travel agent. The Service Proxy/Gateway is the data entrance point of the 3rd-party service provider and its end users. All the entities shown in Figure 2 are service components that the service agent requires from the network operator. 3rd-party service provider gives its own web site and core value-added services to end users and redirects traffic to these supporting servers upon user request. For the travel reservation, users can go to Flight Reservation, Hotel Reservation, or Car Rental Reservation directly. Users can also go to a Deal Search Service to find the best available deal and then make reservations. After making the reservation, the billing is handled by Payment Processing and the Shipping Service takes care of the mailing of receipts, tickets, and other materials. To increase profit, the 3rd-party service provider has a contract with Shopping Sites and attracts users to these shopping sites to purchase travel-related merchandize such as sports gear, books, and souvenirs. To increase the user base, the 3rd-party service provider utilizes a Unified AAA Service to provide a single sign-on experience to end users. Some of the auxiliary services useful for travel arrangements are the Weather Channel and Local News. If some of the news or events are in the form of audio or video, a Media Streaming service could be used. The graph of web service composition just shown has two

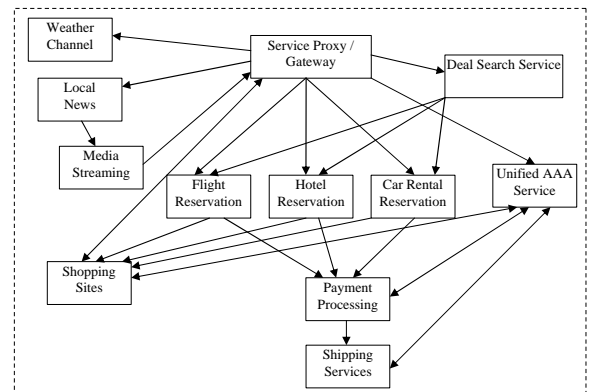


Fig. 2. Service Composition Graph of a Travel Agent.

major differences compared to the service composition model described in the related work.

Firstly, the topology of the web service composition graph is too complicated for existing models to readily handle. In many cases the composition model has to be expressed as a directed cyclic graph. As exemplified in Fig. 2, the proxy

server, news, and streaming form a circle. The Proxy server, flight reservation, Payment processing and Shipping blocks form a service chain while flight reservation, rental reservation and hotel reservation form a parallel architecture.

Secondly, the functionality and QoS metric can not be accurately defined for the service nodes and service links in the graph. This is because each service component might be another service provider with very complicated service logic. For reasons such as security, most of the service logic will not be exposed and the service can only be accessed through a well-defined API. Fulfilling a service request might require multiple back and forth messages between neighboring nodes. Hence, the functionalities of each service component cannot be easily depicted (in many cases requiring run-time user input) and the QoS of each service component and accordingly the end-to-end QoS is hard to defined at setup time.

As a result, the links defined in this graph are no longer the logic interaction between two nodes. Instead, it shows there is noticeable traffic flowing from the source node to destination nodes. The traffic includes all the messages of one conversation and also is the aggregate message traffic of all users using the services.

## B. System Architecture

Figure 3 shows the basic system architecture of the service composition within an operator's network. It includes both an operator's network and its interacting peers outside of its trusted domain. A network operator deploys the *Service Composition Engine* within its trusted domain to manage the service composition process. The *Service Composition Engine* obtains currently available services both within and outside the networks from the *Service Registry*. The *Service Registry* stores the information of service type, characteristics, access information, and other related details and supports different kinds of queries. For example, one Service Registry technology, UDDI [8], can find registry entries that satisfy search criteria such as a particular business entity, a particular service type, access information, and semantic details contained in the model. *Network management* monitors the bandwidth usage of each link and computation capacity of each *application servers/routers* within the network. With all the information, the *service composition engine* calculates the optimal server assignment and sends the results to Service Configuration to configure the involved application servers. The configuration includes a service identifier to uniquely identify a composite service and the addresses of adjacent application servers to forward data traffic of users using this composite service.

The service used for composition can be provided by internal application servers owned and operated by the network operator or by external application servers owned and operated by 3rd parties outside of the trusted space but accessible through well-defined interfaces. To interact with 3rd-party application server, an application proxy is used by the network operator to take care of the message transformation, asynchronous and synchronous transformation, AAA and other coordination functionalities.

3rd-party service providers offer services to end users. To build the services, the 3rd-party service provider decides which parts of the service logic will be implemented locally and what should be supported by the network operators. Before making such a decision, 3rd-party service providers might send inquiries to the *Service Registry* to ask for information of

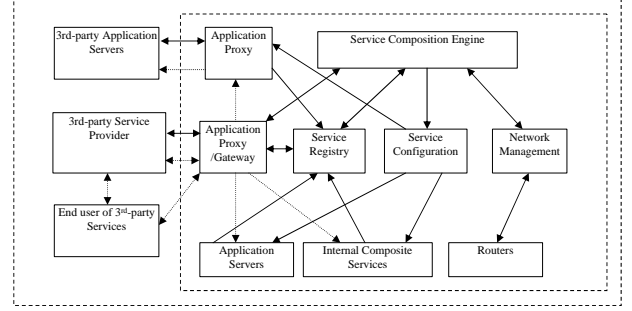


Fig. 3. System Topology of Service Composition.

available services within the network. With this information, 3rd-party service provider can then choose the set of service components that will be supported by the network operator. Once this decision is made, the service provider will sign a Service Level Agreement (SLA) with the network operator to decide the QoS metrics including the total number of users and total bandwidth. With this information, the 3rd-party service provider is able to produce the service composition graph and specify the traffic on each link connecting two service components. Such a service composition graph is then sent to the *Service Composition Engine* for the optimal deployment.

After the service graph is optimally deployed, each end user accessing the composite services will be redirected to application servers within the operator's network. Because each composite service is uniquely identified by a tag, application servers are able to distinguish user requests from different composite services and route the request to the subsequent application server. In Fig. 3, solid lines denote signaling flows during the service deployment phase and dashed lines denote data flow when end users request services.

## IV. SERVICE MAPPING PROBLEM DEFINITION

After presenting the framework to provide QoS support for complex service composition, we formally define the "service mapping" problem (SMP) that is studied in detail in the rest of the paper.

Let service graph  $G_s = (V_s, E_s)$  be a directed graph with vertex set  $V_s$  ( $|V_s| = N_s$ ) and edge set  $E_s$  ( $|E_s| = N_e$ ). Each node  $v_s \in V_s$  is a service component that is required by the 3rd-party service provider. Every edge  $e_s(v_i, v_j) \in E_s$  has an associated cost  $c(e_s)$ , which is the estimated traffic from a source node  $v_i$  to a destination node  $v_j$ .

Let  $P_s = (V_p, E_p)$  be a directed graph that describes the network topology of a network operator. Node set  $V_p$  consists of a set of application servers  $V_a$  and intermediate routers  $V_r$ . Each service component  $v_s \in V_s$  in the service graph is supported by a set of application servers  $S(v_s)$  in the network graph.  $S(v_s)$  is a non-empty subset of  $V_a$ . For every pair of application servers in  $V_a$ , a distance function,  $d(v_{a,1}, v_{a,2})$ , is defined, which is the hop distance of the physical path in the network graph  $P_s$  from source node  $v_{a,1}$  to destination node  $v_{a,2}$ .

The aggregate bandwidth utilization  $C(G_s, P_s, V_a)$  is defined according to the following formula:

$$\sum_{e_s(v_i, v_j) \in E_s} c(e_s) \cdot d(f(v_i), f(v_j)) \quad (1)$$

Then the goal of an optimization algorithm for SMP is to define the mapping function  $f: V_s \mapsto V_a$  that uniquely assigns each service component  $v_s$  to an application server  $f(v_s) \in$

$S(v_s)$  that supports  $v_s$  while minimizing  $C(G_s, P_s, V_a)$  within the network operator's network  $P_s$  to deploy the whole service graph  $G_s$ .

In Eqn. 1, for each service link  $e_s(v_i, v_j)$ , the product of the traffic  $c(e_s)$  on this link and the hop distance  $d(f(v_i), f(v_j))$  between corresponding application servers in the network graph  $P_s$  is calculated. This product captures the effect that the traffic from service component  $v_i$  to  $v_j$  will use the resource of multiple links in the underlying network that interconnect two supporting application servers  $f(v_i)$  and  $f(v_j)$ .

Figure 4 shows an example of SMP that maps a service graph to the underlying physical network and minimizes aggregate bandwidth utilization

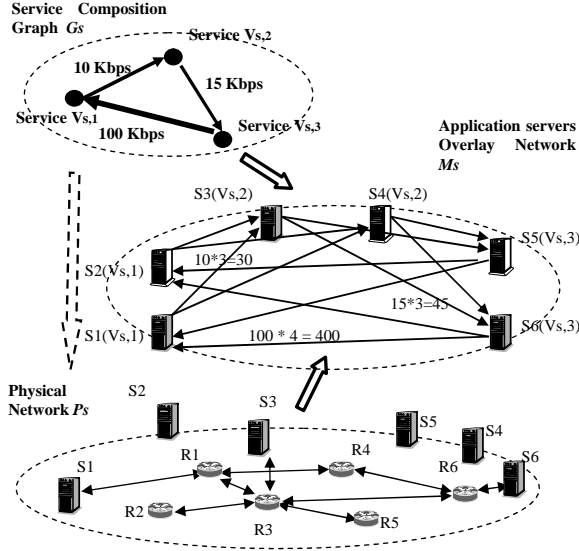


Fig. 4. Example of service mapping problem

There are three different service components,  $V_{s,1}$ ,  $V_{s,2}$ , and  $V_{s,3}$  required from the 3rd-party service provider. The service graph formed by  $V_{s,1}$ ,  $V_{s,2}$ , and  $V_{s,3}$  is a circle. The weights of links among  $V_{s,1}$ ,  $V_{s,2}$ , and  $V_{s,3}$  show the estimated traffic flowing among them. There are six application servers in the physical network supporting these service components. Server  $s_1$  and  $s_2$  support service  $V_{s,1}$ ;  $s_3$  and  $s_4$  support service  $V_{s,2}$ ;  $s_5$  and  $s_6$  support service  $V_{s,3}$ . These servers are distributed at different physical network locations and connected with each other through intermediate routers. For example, the link from  $s_1$  to  $s_3$  has the shortest path of length 3. Then the optimization problem is to choose an application server among  $s_1, s_2$  for  $V_{s,1}$ , a server among  $s_3, s_4$  for  $V_{s,2}$ , and a server  $s_5, s_6$  for  $V_{s,3}$  such that the aggregate bandwidth-distance product for the chosen set of application servers is minimized. If servers  $s_1, s_3,$  and  $s_6$  are chosen, then the aggregate bandwidth-distance product for this set of servers is 475kbps.

Observe that the optimal set of servers can be found by considering all possible combination of servers, but this approach has exponential time complexity. Given a generic service graph with arbitrary topology and traffic requirement and a network graph having a number of servers for each service component, it is NP-hard to compute the optimal set of servers, as we will show, and therefore unlikely to be computable in polynomial time. Of course, for specific classes of graphs, one may be able to obtain more efficient solutions.

#### A. Integer programming formulations

To develop exact algorithm for the SMP defined in Eqn. 1, we formally formulate it as an integer programming (IP)

problem so that standard linear programming and integer programming techniques can be used to find the optimal solution.

In order to formulate the problem, we compose an overlay network of application servers  $M_s = (V_m, E_m)$  from the given service graph  $G_s$  and network graph  $P_s$ . The node set  $V_m$  of  $M_s$  consists of all the application servers that support a service component of  $G_s$ . So  $V_m$  is equal to  $V_a$ . A link  $e_m \in E_m$  is defined from node  $s_i(v_i)$  to  $s_j(v_j)$  if and only if there is a service link  $e_s$  in graph  $G_s$  from service component  $v_i$  to  $v_j$ . For each link  $e_m$  from node  $s_i(v_i)$  to  $s_j(v_j)$  we define a weight function  $c(e_m) = c(e_s(v_i, v_j)) \cdot d(s_i, s_j)$ ; i.e., the bandwidth-distance product of  $e_m$ . An example of composing overlay network  $M_s$  is also shown in Figure 4. Here we assume that each application server supports only one service component. Later we will describe the process to decompose a server supporting multiple service components into multiple servers supporting only one service component each. So techniques described here are still feasible.

The integer programming formulation of the SMP is then based on the composed overlay network  $M_s$ . We define for each server  $s_m(v_i)$ , a variable  $y_{s_m}$  which is equal to one if server  $s_m$  is chosen to support service component  $v_i$ , and zero otherwise. We define for each  $e_m(s_i(v_i), s_j(v_j)) \in E_m$ , a variable  $x_{e_m}$  which is equal to one if edge  $e_m$  is included in the mapping, and zero otherwise.

We further consider three additional constraints imposed by the service mapping problem. For each  $v_i$ , only one application server is chosen from the set,  $S(v_i)$ , of all servers supporting  $v_i$ . So we introduce the constraint

$$\sum_{s_m \in S(v_i)} y_{s_m} = 1, \quad \forall v_i \in G_s. \quad (2)$$

Let  $E(i, j)$  be the set of all the links in  $M_s$  that point from servers supporting service  $v_i$  to servers supporting service  $v_j$ . Because there is only one link chosen in  $E(i, j)$  that goes from a server chosen to support  $v_i$  to another server chosen to support  $v_j$ , we introduce the constraint

$$\sum_{e_m \in E(i, j)} x_{e_m} = 1, \quad \forall v_i, v_j \in V_s, v_i \neq v_j. \quad (3)$$

Furthermore, if an application server  $s_m$  is chosen to support a service component  $v_i$ , all traffic incident to the service component  $v_i$  must enter or leave the same server  $s_m$ . So, all the links incident to the chosen server are included in the mapping and none of the links of other servers supporting  $v_i$  are included. Let  $D(v_i)$  be the sum of the in-degree and out-degree of a service component  $v_i$  in  $G_s$  and  $E(s_m)$  be the set of all links in  $M_s$  that enter or leave  $s_m$ . We introduce the constraint

$$\sum_{e_m \in E(s_m)} x_{e_m} = D(v_i) \cdot y_{s_m}, \quad \forall s_m \in V_m. \quad (4)$$

Combining these constraints, we obtain the following integer programming formulation of the SMP:

$$\begin{aligned} & \text{minimize} && \sum_{e_m \in E_m} c(e_m) x_{e_m} \\ & \text{subject to} && \sum_{s_m \in S(v_i)} y_{s_m} = 1, && \forall v_i \in G_s \\ & && \sum_{e_m \in E(i, j)} x_{e_m} = 1, && v_i \neq v_j, \forall v_i, v_j \in V_s \\ & && \sum_{e_m \in E(s_m)} x_{e_m} = D(v_i) y_{s_m}, && \forall s_m \in V_m \\ & && x_{e_m} \in \{0, 1\}, \\ & && y_{s_m} \in \{0, 1\}. \end{aligned} \quad (5)$$

After SMP is formulated as a standard integer programming problem, techniques such as branch and bound [9] can be used to derive the optimal value. Branch and bound divides original feasible solution set into many smaller feasible solution sets and then solves subproblems on these smaller sets. During the calculation, it uses bounds on the optimal cost to avoid exploring certain subproblems. A popular method to obtain such a bound is to use the optimal cost of the linear programming relaxation.

Typically techniques such as branch and bound have better performance than brute force search. However, there are  $O(V_a^2)$  variables  $x_{e_m}$  and  $O(V_a)$  variables  $y_{s_m}$  in Eqn. 5. So in worst case, branch and bound has to calculate  $O(2^{n^2})$  subproblems and the time complexity is still exponential.

In the following section, we will prove that the SMP is indeed NP-hard. Accordingly we will propose two approximation algorithms and prove their approximation ratio.

## V. NP-HARDNESS OF SERVICE MAPPING PROBLEM

The time required to solve the SMP depends on the topologies of service graph  $G_s$  and the network graph  $P_s$  and the number of application servers for each service component. For some special cases, SMP can be solved efficiently. For example, if there is only one application server for each service component, then no matter how many service components  $G_s$  has, the total number of server combination is one and SMP is solved in constant time. Another example is the simple chain topology which can be handled in polynomial time by modifying Dijkstra's shortest path algorithm [3].

However, a brute-force algorithm requires super polynomial time in general. For example, for a service graph with  $k$  service components and a physical network that has 3 servers for each service, the total number of server combination is  $3^k$ , which is exponential in  $k$ .

In this section, we will prove the NP-hardness of SMP by reducing the well-known NP-complete problem 3-CNF-SAT to it (3-CNF-SAT was proved NP-hard by Cook [10]). We then show, the problem is hard, even to approximate, within a constant factor.

The 3-CNF-SAT has following definition. A Boolean formula  $\phi$  is in 3-conjunctive normal form, or 3-CNF, if  $\phi = C_1 \wedge \dots \wedge C_k$ , where each  $C_i$  is the disjunction of exactly three literals. A literal in the formula is an occurrence of a variable or its negation. Assume that there are  $n$  variables total; call these  $x_1, \dots, x_n$ . For example, the Boolean formula

$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4) \quad (6)$$

is in 3-CNF. The 3-CNF-SAT problem is to find out whether a given Boolean formula  $\phi$  is satisfiable; i.e., there is an assignment of the variables for which the formula evaluates to true.

We focus on the decision version  $\langle G_s, G_p, c \rangle$  of the SMP, which is described as follows: can we choose one server (among all servers in  $G_p$ ) for each service component in  $G_s$  such that the sum of bandwidth-distance product of all links among these chosen servers in  $G_p$  is at most  $c$ . The NP-hardness of the optimization version of SMP follows immediately from the NP-completeness version of the decision version.

*Theorem 1:* The Decision Service Mapping Problem (DSMP) is NP-Complete.

*Proof:* It is easy to see that DSMP  $\in NP$ . Given an instance of decision problem  $\langle G_s, G_p, c \rangle$  that has total  $\ell = |V_a|$  servers, it takes  $O(\ell^2)$  to sum the bandwidth-distance product of all links among these servers and decide whether the sum is bounded by  $c$ .

We now show NP-hardness of DSMP by reducing 3-CNF-SAT to it. Given the instance of 3-CNF  $\phi = C_1 \wedge \dots \wedge C_k$  with  $k$  clause, we convert it to an application overlay network  $M_s$  of SMP as follows. The graph  $M_s$  has  $k$  sets of 3 vertices. Call these sets  $S(v_1), \dots, S(v_k)$ . Each  $S(v_i)$  contains 3 vertices (corresponding to the literals in  $i^{th}$  clause) where all 3 vertices in  $S(v_i)$  support the same service component  $v_i$ . Now, we place a directed from each vertex in each  $S(v_i)$  to each vertex in  $S(v_j)$ ,  $i \neq j$ . The weight of the directed edge is  $\alpha$  if two vertices of the edge *do not* correspond to literals that are the negation of each other and is  $\beta$  otherwise (e.g., the edge weight is  $\beta$  on the edge between vertices corresponding to  $x_1$  and  $\neg x_1$ ). Further, we pick  $\alpha < \beta$ .

Hence, the generated  $M_s$  has three characteristics. First, the service graph  $G_s$  is a clique. Second, the bandwidth-delay product of each link is either  $\alpha$  or  $\beta$ . Third, there are exactly 3 servers supporting each service component in  $G_s$ .

Now, we claim that the original formula  $\phi$  is satisfiable iff the resulting SMP has an aggregate bandwidth-distance product of  $n(n-1)\alpha$ . First, suppose the original formula is satisfiable. Then, there is a satisfying assignment to the literals  $x_1, \dots, x_n$ . For each clause, there must be one literal whose value is true within that clause. Pick one such true literal from each clause. Consider the vertices  $s_i(v_i)$  corresponding to the true literal in clause  $C_i$  belonging in set  $S(v_i)$  in  $M_s$ , for  $1 \leq i \leq k$ . Now, this collection of vertices forms a clique (since there is an edge between every vertex in one clause and every vertex in another clause). Moreover, because in two clauses both the literals evaluate to true, they cannot be negations of each other. Therefore, each links has a bandwidth-distance product of  $\alpha$ . We have  $n(n-1)$  edges (since we have  $n(n-1)/2$  pairs and there is one edge in each direction). The sum of these values is  $n(n-1)\alpha$ . This provides a proof for one direction.

Now, let us assume that  $M_s$  has a service assignment of weight  $n(n-1)\alpha$ . In this case, consider the vertices in the clique. The literals corresponding to these vertices can be set to true. This gives a satisfying assignment to each clause. Moreover, we have no contradictions since the clique weight is  $n(n-1)\alpha$ , but the weight between any literal and its negation in another clause would be  $\beta$ ; therefore the edge is not in the clique since its inclusion would yield a weight greater than  $n(n-1)\alpha$  since  $\beta > \alpha$ .

Therefore, the algorithm for converting the formula to  $M_s$  constitutes a reduction. Finally, it is not hard to see that this reduction can be computed in polynomial time. This follows since the graph  $M_s$  has  $3k$  vertices. So we can initialize the adjacency matrix in at most  $9k^2$  steps. Filling it is straightforward and can be done with a single scan of the formula. Thus the whole reduction is polynomial time.  $\square$

The reduction is depicted in Fig. 5 using Eqn. 6 as an example.

Now, let us examine what this reduction yields with respect to approximation ratios. Suppose that there is a polynomial-time algorithm that always performs within an approximation factor of  $\hat{b}$ . We claim that the existence of such an algorithm would imply  $P = NP$ . The reasoning is simple – if we construct the graph used in the above reduction where  $\beta$  is

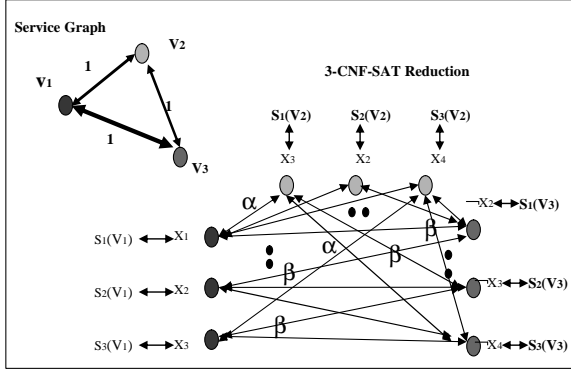


Fig. 5. Reduction of 3-CNF-SAT in Equation 6

especially computed to be sufficiently larger than  $\alpha$ , then the gap between the optimal solution and any suboptimal solution would be large. Therefore, any algorithm performing within a given approximation bound would necessarily pick the optimal solution. However, the optimal solution results in a satisfying assignment for the formula  $\phi$  on which the graph is based. Specifically, if we set

$$\beta > \frac{1}{2} \left( \hat{b}(n(n-1)\alpha) - (n^2 - n - 2)\alpha \right), \quad (7)$$

then this is sufficient. Here's why. The optimal algorithm would yield a solution of weight  $n(n-1)\alpha$ . Any "suboptimal" solution must have at least one  $\beta$  edge (in each direction). Therefore, a suboptimal solution has weight at least  $(n(n-1) - 2)\alpha + 2\beta$ . And the approximation ratio of such a suboptimal solution is:

$$\frac{(n(n-1) - 2)\alpha + 2\beta}{n(n-1)\alpha}.$$

Now, if the above quantity is greater than  $\hat{b}$ , then the supposed approximation algorithm must pick the optimal solution (otherwise it failed to meet its claimed approximation ratio). Setting  $\beta$  according to eqn. 7 is sufficient. Therefore, we have proved:

**Theorem 2:** For any constant  $\hat{b}$ , there is no polynomial-time algorithm for SMP with approximation  $\hat{b}$  unless  $P = NP$ .

## VI. APPROXIMATION ALGORITHMS

In this section, we introduce two polynomial time "heuristic" approximation algorithms that provide a feasible, but suboptimal, solutions to SMP. We also prove a generic (non-constant factor) upper bound  $\hat{\alpha}$  on the approximation ratio of any algorithm that yields a feasible solution. We further prove that in the worse case the approximation ratio of both approximation algorithms are equal to  $\hat{\alpha}$ . Our simulations later show that both algorithm perform very well in reality. For example, in our simulation, when  $\hat{\alpha}$  is 12, both algorithm has approximation ratio less than 2 for the types of graphs encountered in practice.

Given a service graph  $G_s$ , a network graph  $P_s$ , and a set of application servers  $V_a$  in  $P_s$ , we form the overlay network of application server  $M_s$ . Let  $N_e$  be the number of links in  $G_s$ ,  $C_{max}$  be the sum of the  $N_e$  highest bandwidth-distance product of all links in  $M_s$ , and  $C_{min}$  be the sum of the  $N_e$  lowest bandwidth-distance product of all links in  $M_s$ .

**Lemma 1:** Any feasible SMP algorithm has an approximation ratio of at most  $\hat{\alpha} = \frac{C_{max}}{C_{min}}$ .

**Proof:** The optimal value,  $C$ , of SMP is bounded from below by  $C_{min}$  because any feasible solution of SMP is no less than

$C_{min}$ . The aggregate bandwidth-distance product,  $C_{approx}$ , of any approximation algorithm's solution is no bigger than  $C_{max}$ . So the approximation ratio  $\alpha \doteq \frac{C_{approx}}{C} \leq \frac{C_{max}}{C_{min}} \doteq \hat{\alpha}$ .  $\square$

### A. Minimum-weight Heuristic Approximation Algorithm

The Minimum-weight approximation algorithm (MW) discussed in this section and the longest chain approximation algorithm discussed in the following section are both greedy. They differ from each other in the order they organize and search service components in  $G_s$ .

The search order of MW is based on the observation that many service graphs have a parallel fan-out architecture. One example of such a fan-out architecture is the service graph of a travel agent shown in Figure 2. Part of this figure shows traffic of the travel agent's users flows through "service proxy/gateway" to services of "flight reservation", "hotel reservation", and "car rental" in parallel. And then the billing traffic from these services flow to a "payment processing" service in parallel as well.

To take advantage of such an inherent fan-out architecture in many complex service graphs, MW first runs breadth-first search (BFS) in a service graph and uses the resulting BFS tree to arrange all service components. The root of the BFS tree is the service proxy/gateway with depth 0. The leaf nodes will have the largest depth. Because the purpose of BFS is to arrange service nodes according to their distance from the root, BFS uses hop distance as the metric.

Then MW begins its greedy procedure from the leaf nodes to the root node in decreasing order of nodes' depth. For each service component  $v_i$ , each of its supporting servers  $s_m(v_i) \in S(v_i)$  calculates a local weight function  $w(s_m(v_i))$ . The server with the lowest weight  $w(s_m(v_i))$  is chosen to support service component  $v_i$ . This procedure is repeated until all service components have been assigned to a server.

The local weight function  $w(s_m(v_i))$  is calculated as follows. At the beginning of the iteration, MW initializes the weight function  $w(s_m)$  of each server to zero. Then, for a chosen service component  $v_i$ , the local weight of its supporting server is calculated using the formula:

$$w(s_m(v_i)) = \sum_{v_j \in \text{adj}e(v_i)} c(e_s) \cdot d(s_m(v_i), f(v_j)). \quad (8)$$

In Equation 8,  $\text{adj}e(v_i) = \{v_j | v_j \in V_s, e_s(v_i, v_j) \in E_s\}$  is the set of neighboring nodes of  $v_i$  in  $G_s$ . We let  $c(e_s)$  denote the estimated traffic on the service link  $e_s$  and let  $d(s_m(v_i), f(v_j))$  denote the hop distance of the route from application server  $s_m(v_i)$  to the application server  $f(v_j)$  supporting service component  $v_j$ . If  $f(v_j)$  has not been assigned yet, the shortest distance from  $s_m(v_i)$  to any server supporting  $v_j$  is chosen as  $d(s_m(v_i), f(v_j))$ . The pseudocode of the MW is shown in Fig. 6.

Fig. 7 shows an example of using the MW to solve SMP. There are 5 service components in the service graph. For each service component, there are two application servers supporting it; namely, server  $i$  and  $j$  for  $V_{s1}$ , server  $e$  and  $f$  for  $V_{s2}$ , server  $g$  and  $h$  for  $V_{s3}$ , server  $c$  and  $d$  for  $V_{s4}$ , and server  $a$  and  $b$  for  $V_{s5}$ . All links between two application servers have length 2, except from  $e$  to  $a$  and from  $c$  to  $e$ , which we assume have distance 1. Fig. 7 shows the results of BFS search and local weight calculation of each application

Minimum-weight Algorithm( Input:  $G_s, P_s, V_a$ )

```

{
  Run BFS search in  $G_s$ ;
  Order nodes of  $G_s$  in the decreasing order of depth;
  Set weight  $w$  of each server  $sm$  to 0;
  foreach (  $v_i$  in  $G_s$  ) {
    foreach (  $sm$  in  $S(v_i)$  )
      calculate weight  $w(sm)$ ;
    assign  $sm$  with minimum weight to support  $v_i$ ;
  }
}

```

Fig. 6. Pseudocode of Minimum-weight algorithm

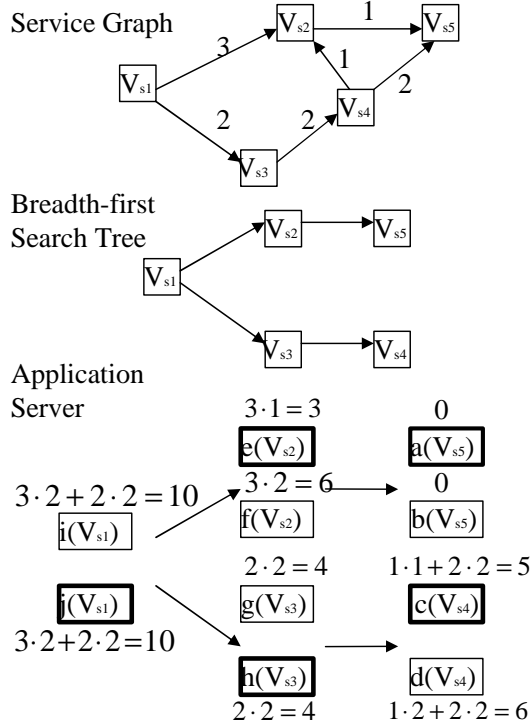


Fig. 7. Example of minimum-weight algorithm

server. The chosen application server is marked with thicker box. We break ties randomly.

Since the BFS search in  $G_s$  takes  $O(V_s + E_s)$  time and the calculation of local weights of servers in  $V_a$  takes at most  $O(|V_a|^2)$  time, the time complexity of MW is polynomial. The approximation ratio of MW is given in the following theorem.

*Theorem 3:* MW has an approximation ratio of  $\hat{\alpha}$ .

*Proof:* Lemma 1 shows that  $\hat{\alpha}$  is an upper bound for any approximation ratio. So, we only need to prove that  $\hat{\alpha}$  is reachable for MW.

Consider the instance shown in Fig.8. The service graph  $G_s$  is a chain. There is one server ( $a$ ) for  $v_{s1}$ , two servers ( $b$  and  $c$ ) for  $v_{s2}$ , two servers ( $d$  and  $e$ ) for  $v_{s3}$ , and one server ( $f$ ) for  $v_{s4}$ . The composed overlay network  $M_s$  has the link weights shown in the figure. It is easy to see that the optimal server assignment is  $\{a, c, e, f\}$ , which has bandwidth-distance product  $m$ . MW chooses  $\{a, b, d, f\}$ , which has bandwidth-distance product  $3(m-1)$ . The approximation ratio for this scenario is  $\alpha = \frac{3(m-1)}{m}$  and the upper bound is  $\hat{\alpha} = \frac{2m+(m-1)}{m-1}$ . As  $m$  goes to  $\infty$ ,  $\alpha$  approaches  $\hat{\alpha}$ .  $\square$

Notice that Theorem 3 provides a worst-case approximation ratio. Our simulation results later show that MW performs very

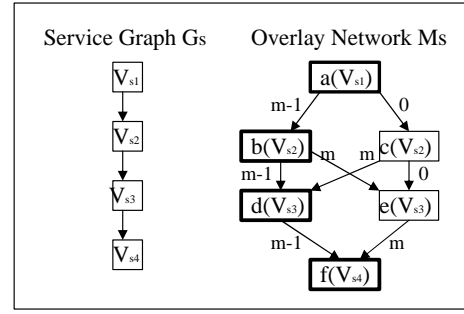


Fig. 8. The worst case of minimum weight algorithm

well in most of the scenarios.

### B. The longest chain heuristic approximation algorithm

The previous section discussed the MW which targets service graphs with parallel fan-out architecture. In this section we present another greedy algorithm, the longest chain approximation algorithm (LC), that is geared towards service graphs with more sequential architecture. The service graph discussed earlier is an example of this architecture.

To find the inherent sequential architecture in a complex service graph, LC first runs depth-first-search (DFS) in the service graph. Because DFS can produce several disconnected DFS trees, LC manipulates these DFS trees one at a time.

For each DFS tree, the longest service path is from the root to the leaf with the largest depth. Then, the second longest chain in this DFS tree is found whose source node is on the first service chain and whose destination node is another leaf. This procedure continues until all service nodes in this DFS tree are covered by some service chain. After service chains have been found in all DFS trees, they are ordered from the longest to the shortest. Ties are broken randomly.

Longest-Chain Algorithm( Input:  $G_s, P_s, V_a$ )

```

{
  Run DFS search in  $G_s$ ;
  foreach ( DFS tree )
    find all service chains in the tree;
  Order service chains  $SC_k$  in the decreasing order of length;
  Set weight  $w$  of each server  $sm$  to 0;
  foreach (  $SC_k$  ) {
    form overlay network  $M_s(SC_k)$ ;
    foreach (  $sm$  in  $M_s(SC_k)$  )
      calculate weight  $w(sm)$ ;
    foreach ( pair of root server and leaf server )
      calculate the shortest path;
    choose the path  $P_i$  with lowest weight;
    assign all  $sm(v_i)$  on path  $P_i$  to support  $v_i$ ;
  }
}

```

Fig. 9. Pseudocode of the longest chain algorithm

Then, beginning with the longest service chain, LC decides the mapping of service components on each service chain. For each service chain  $SC_k$ , LC first composes the overlay network  $M_s(SC_k)$  of application servers using  $SC_i$  as the service graph. Then each server  $s_m$  uses Eqn.9 described below to calculate the local weight  $w(s_m)$ . LC then runs the shortest path algorithm in  $M_s(SC_k)$  from application servers supporting root service component to application servers supporting the leaf service component. The path with the lowest weight is chosen and servers on this path are selected to support corresponding service components. Finally, LC decides whether all service chains have been calculated; if so, it terminates. The pseudocode of LC is shown in Fig. 9.

The local weight calculation in LC is almost the same as the calculation in MW. The only difference is that the bandwidth-distance product of the service link from a server to its immediate child in the service chain  $SC_k$  is excluded from the weight calculation. This is because the bandwidth-distance product of this link has already been included in the calculation of the shortest path. Hence, the local weight calculation of LC is:

$$w(s_m(v_i)) = \sum_{v_j \in \text{adj}_{e_{sc}}(v_i)} c(e_s) \cdot d(s_m(v_i), f(v_j)) \quad (9)$$

and  $\text{adj}_{e_{sc}}(v_i) = \{v_j | v_j \in V_s, e_s(v_i, v_j) \in E_s, e_s(v_i, v_j) \notin SC_k\}$ .

The weight of a path of service chain  $SC_k$  is calculated as follows:

$$\sum_{e_s(v_i, v_j) \in SC_k} c(e_s) \cdot d(f(v_i), f(v_j)) + \sum_{v_i \in SC_k} w(f(v_i)). \quad (10)$$

Fig. 10 shows an example of using LC to solve SMP.  $G_s$ ,  $P_s$ , and  $V_a$  are the same as those in Fig.7. The DFS produces one DFS tree. In this tree, the longest chain is  $V_{s1} \rightarrow V_{s2} \rightarrow V_{s3}$  of length 3. The second chain is  $V_{s1} \rightarrow V_{s3} \rightarrow V_{s4}$ . Since the root of the second chain is already included in the first chain, the length of the second chain is 2. The weight calculation of each application server is shown in the example. Also shown is the calculation of shortest path according to Eqn. 10. After application servers on the first service chain are chosen, the application servers on the second service chain are chosen in the same way. The chosen application servers are denoted with a solid box.

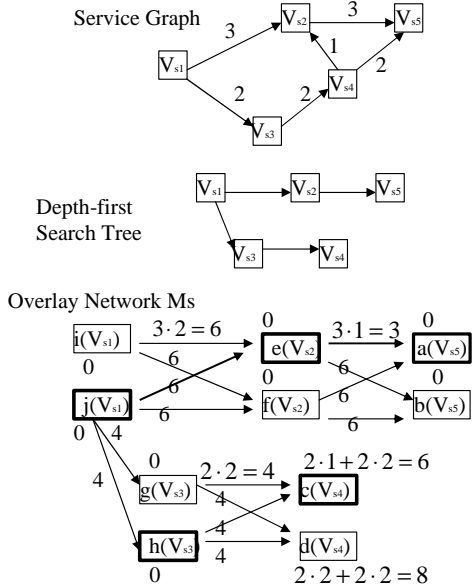


Fig. 10. Example of the "longest service chain" approximation problem

LC runs in polynomial time since each step of LC finishes in polynomial time. The computation of LC is much more complex than MW. However, its time complexity is greater than that of MW. In the following, we prove the approximation ratio of LC, however, is the same as MW in the worst case.

*Theorem 4:* LC has approximation ratio  $\hat{\alpha}$ .

*Proof:* Lemma 1 has shown that  $\hat{\alpha}$  is the upper bound for any approximation ratio. So, we only need to prove that  $\hat{\alpha}$  is reachable for LC.

Consider the instance shown in Fig.11. Running DFS in  $G_s$  produces two chains,  $V_{s1} \rightarrow V_{s2} \rightarrow V_{s3}$  and  $V_{s1} \rightarrow V_{s4} \rightarrow V_{s5}$ . There is one server ( $a$ ) for  $v_{s1}$ , two servers ( $b$  and  $c$ ) for  $v_{s2}$ , two servers ( $d$  and  $e$ ) for  $v_{s3}$ , one server ( $f$ ) for  $v_{s4}$ , and one server ( $g$ ) for  $v_{s5}$ . The composed overlay network  $M_s$  has the link weights shown in the figure. It is easy to check that optimal server assignment is  $\{a, c, e, f, g\}$ , which has bandwidth-distance product  $4m$ . LC chooses  $\{a, b, d, f, g\}$ , which has bandwidth-distance product  $6m - 2$ . The approximation ratio for this scenario is  $\alpha = \frac{6m-2}{4m}$  and the upper bound is  $\hat{\alpha} = \frac{6m}{4m-2}$ . As  $m$  goes to  $\infty$ ,  $\alpha$  approaches  $\hat{\alpha}$ .  $\square$

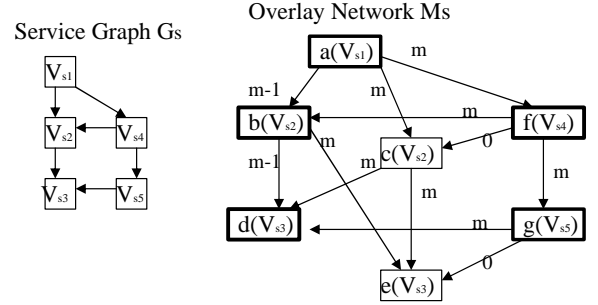


Fig. 11. The worst case of the longest chain algorithm

### C. Local search algorithms

In this section, we will present several local search algorithms that further improve the performance of MW and LC. The main idea of these local search algorithms is to start at the assignment  $f(G_s)$  produced by MW and LC, and then evaluate the aggregate bandwidth-distance product  $C$  for some other assignment  $f'(G_s)$ , which is a "neighbor" of  $f(G_s)$ . If a neighbor  $f'(G_s)$  with lower  $C$  is found, select  $f'(G_s)$  and repeat the same process. If no such neighbor is found, the process stops: a *local optimum* has been found.

There is a generic tradeoff in such local search algorithms. When we consider larger neighborhoods, there are few local minima and a better solution is likely to be found when the algorithm terminates. On the other hand, the algorithm is slower since more feasible solutions have to be compared.

The three local search algorithms presented here follow the same logic and only differ in the way they define "neighborhood". *1OPT* local search considers assignments  $f(G_s)$  and  $f'(G_s)$  neighbors if we can obtain one from the other by changing the assignment of one service component. Similarly, *2OPT* and *3OPT* local search considers  $f(G_s)$  and  $f'(G_s)$  neighbors if we can obtain one from the other by concurrently changing the assignment of two or three service components respectively.

## VII. PERFORMANCE EVALUATION

We evaluate the performance of MW and LC in this section. We also present the results of how three local search algorithms can improve the performance of MW and LC. Service graphs with different characteristics are used to determine which approximation algorithm is more suitable for a given graph. The results are further compared with the optimal solution to determine the approximation ratio of two algorithms.

### A. Simulation Setup

The simulations are conducted using the physical topology graph produced by the Georgia Tech Internet Topology Models

(GT-ITM) [11]. The simulation results presented here use a random physical graph  $P_s$  that has 100 nodes and 354 links. The largest hop distance between two servers in  $P_s$  is 7. Two service graphs,  $G_{s1}$  and  $G_{s2}$ , shown in Fig. 12 are considered. Both service graphs are complicated. Fig. 12.a has more sequential characteristics while Fig. 12.b has more parallel fan-out characteristics. As shown later in the results section, the topology of service graphs has great impact on the performance of both approximation algorithms.

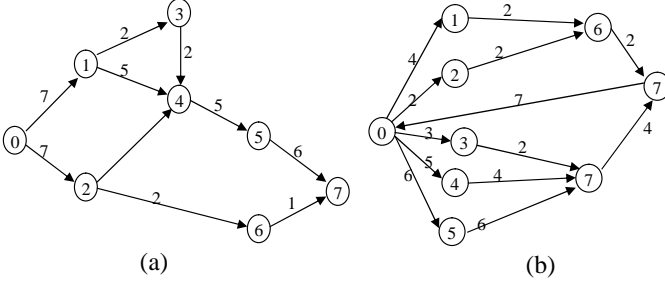


Fig. 12. Service Graph  $G_{s1}$  and  $G_{s2}$

For each service component in  $G_s$ , there is at least one application servers in  $P_s$  supporting it. The simulation is carried on as follows. At each round,  $i$  application servers are chosen randomly in  $P_s$  to support each service component in  $G_s$ . Then, MW and LC are used respectively to assign each service component to one application server. To make comparison, the optimal assignment is also calculated using the exact algorithms described in section IV. For each  $i$ , 20 rounds are repeated and the average is calculated. This process is repeated for each  $i$  between 1 and 9. Three local search algorithms,  $1OPT$ ,  $2OPT$ , and  $3OPT$ , are then used to improve the results of MW and LC.

## B. Simulation Results

The first set of results, Fig. 13, shows the simulation results of service graph  $G_{s1}$  in Fig. 12.a. When each service component has only one supporting application server, both MW and LC calculate the same result as the optimal algorithm. As the number of supporting application servers for each service component increases, the difference between both approximation algorithms and the optimal algorithm also increases. When the number of server per service is 9, MW performs 35% on average (80% maximum and 5% minimum) worse than optimal algorithm and the LC performs 15% on average (47% maximum and 0% minimum) worse than the optimal algorithm. More clearly shown in Fig.14, LC performs better than the MW for service graph  $G_{s1}$ .

The second set of results shows the performance of two algorithms for service graph  $G_{s2}$ . Fig. 15 presents the main simulation results of two algorithms. As the number of servers per service increases, two algorithms' performance deteriorate also compared with optimal value. When the number of servers per service is 9, MW behaves 22% on average worse than optimal value and LC behaves 25% worse than the optimal value. Comparing the results of Fig.14 and Fig. 15, we can observe that the topology of the service graph has big impact on the performance the algorithms. Both algorithms perform better for  $G_{s2}$  than  $G_{s1}$ . Furthermore, MW performs worse than LC in  $G_{s1}$  but better in  $G_{s2}$ . This shows that both algorithms are suitable for different service graphs. In all cases, the approximate ratio of both algorithms is better than

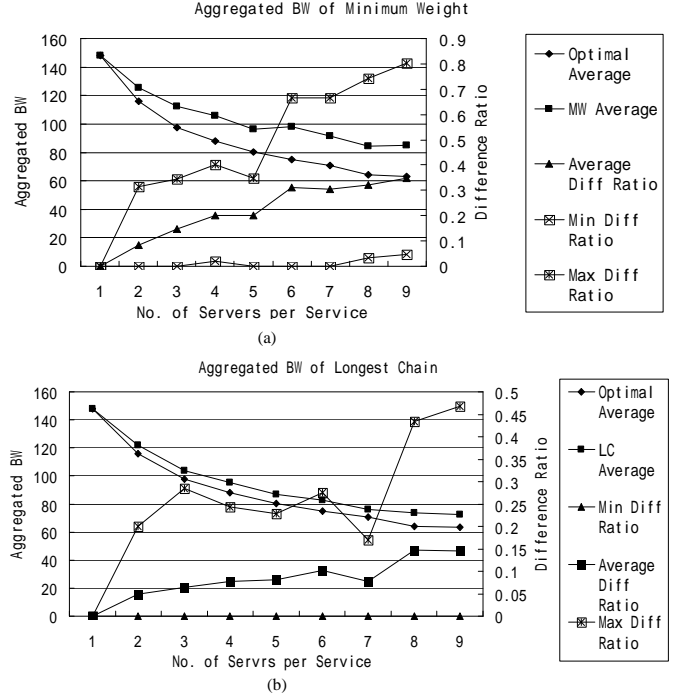


Fig. 13. Aggregate BW of MW and LC

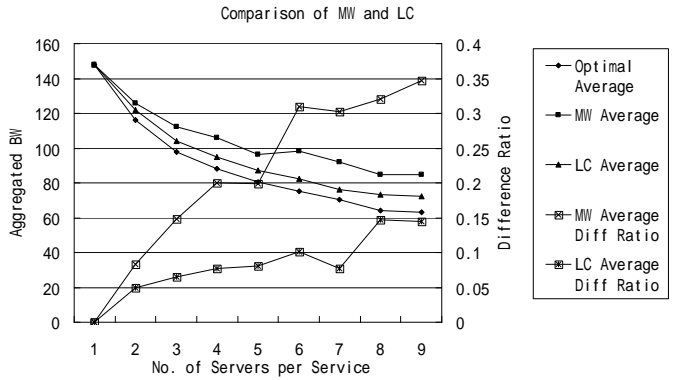


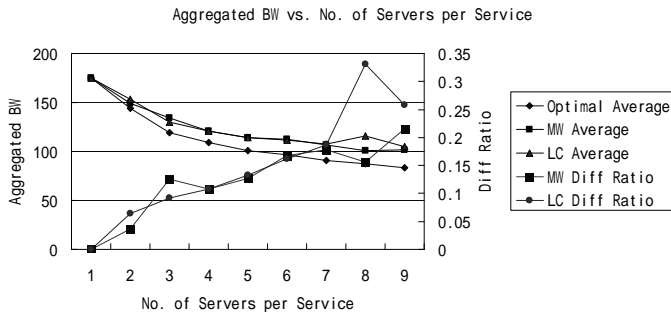
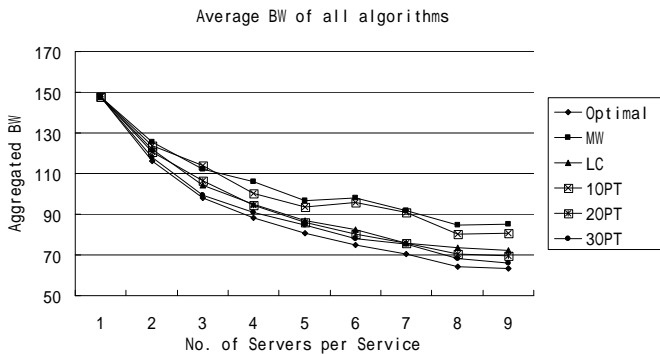
Fig. 14. Comparison between MW and LC for  $G_{s1}$

2. This is much better than the worse case upper bound,  $\hat{\alpha}$ , which is approximately 12.

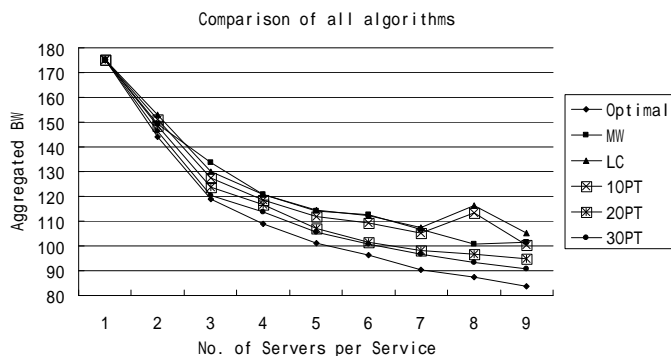
The third set of results shows the performance of local search algorithms. In Fig. 16, algorithms  $1OPT$ ,  $2OPT$  and  $3OPT$  start from the result of MW. Service graph  $G_{s1}$  is used. In Fig. 17, local search algorithms start start from the result of LC. Service graph  $G_{s2}$  is used. Both figures show that  $1OPT$  does not improve the performance much.  $1OPT$  has similar performance to MW in Fig. 16 and to LC in Fig. 17. On the other hand, both  $2OPT$  and  $3OPT$  can improve the performance greatly.  $2OPT$  performs a little bit worse than  $3OPT$  due to the smaller neighborhood size.  $3OPT$ 's performance closely follows the optimal value and the difference slightly increases as the number of servers per service component increases. When the number of servers per service component is 9,  $3OPT$  behaves 4.75% worse than optimal value in Fig. 16 and 8.2% worse than optimal value in Fig. 17.

## VIII. DISCUSSIONS OF SMP

After discussing our main algorithms of the SMP, we briefly describe some techniques here that extend these algorithms to other areas.

Fig. 15. Comparison between MW and LC for  $G_{s0}$ Fig. 16. Performance of local search algorithms for  $G_{s1}$ 

- **Multiple services per server** Network graph  $P_s$  discussed so far assumes that each server only support one service. This is mainly for the convenience of discussion and not a limitation. Note that a server supporting multiple service components can be easily transformed into multiple servers supporting only one service component each. For each newly added server node, links of length 0 are added to connect it to other new servers and links of length 1 are added to connect it to all neighbor nodes of original server. Then, servers in the resulting network graph  $P'_s$  only support one service.
- **Bandwidth limitation** When forming the overlay network of application server  $M_s$ , our algorithms select the shortest distance path as the virtual link connecting two servers. It is assumed that the available bandwidth on this path is bigger than the estimated traffic on the corresponding service link in  $G_s$ . If this is not the case, a pre-processing procedure is required before forming  $M_s$ . In the process, physical links that do not meet the bandwidth requirement are removed before the shortest distance algorithm is used to find the shortest path among two servers. Such path is referred to as a shortest-widest path [12].

Fig. 17. Performance of local search algorithms for  $G_{s2}$ 

- **Server load and network load balancing** Besides minimizing the aggregate bandwidth utilization, our algorithm can also accommodate other QoS goals such as balancing the server load and network load. To achieve this, some weight functions are defined to reflect current server load or network traffic load. Then such weight will be included into the local weight calculation in Eqn. 8 or Eqn. 9 to select appropriate servers or links.

## IX. CONCLUSIONS AND FUTURE WORKS

In this paper, we develop a service composition architecture that optimizes the aggregate bandwidth utilization when deploying a general service composition graph required by a 3rd-party service provider. We formalize the service mapping problem (SMP) as an integer programming problem and prove it to be NP-hard even to approximate within a constant factor. Next, two polynomial-time heuristic approximation algorithms are developed and their approximation ratios are proved. Several local search algorithms are then developed to improve the performance of two approximation algorithms. Finally, our simulations demonstrate the effectiveness of our solution.

The research on supporting QoS for service composition has just begun. Some of our future work in this area include: *a.* develop technologies that can estimate traffic on each service link and automatically produce service composition map; *b.* study the server deployment problem of how to deploy servers into a network given a large number of service composition graphs; and *c.* investigate the issues of combining network oriented QoS optimization and end-to-end user oriented QoS optimization solutions.

## REFERENCES

- [1] W3C. Services Description Language (WSDL) 1.1.
- [2] X. Gu, K. Nahrstedt, R. Chang, and C. Ward. QoS-Assured Service Composition in Managed Service Overlay Networks. In *IEEE ICDCS'2003*, May 2003.
- [3] J. Jin and K. Nahrstedt. QoS Service Routing in One-to-One and One-to-Many Scenarios in Next-Generation Service-Oriented Networks. In *IEEE IPCCC'2004*, Apr. 2004.
- [4] T. Andrews. Specification: Business Process Execution Language for Web Services Version 1.1.
- [5] D. Xu and K. Nahrstedt. Finding service paths in an overlay media service proxy network. In *SPIE/ACE MMCN*, Jan. 2002.
- [6] J. Jin and K. Nahrstedt. Large-scale service overlay networking with distance-based clustering. In *ACM/IFIP/USENIX International Middleware Conference*, Jun. 2003.
- [7] M. Wang, B. Li, and Z. Li. sFlow: Towards resource-efficient and agile service federation in service overlay networks. In *IEEE ICDCS'2004*, Mar. 2003.
- [8] UDDI SPEC TC. UDDI Version 3.0.
- [9] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.
- [10] S. A. Cook. The Complexity of Theorem Proving Procedures. In *Proc. of 3rd ACM STOC*, pages 151–158, 1971.
- [11] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internet network. In *IEEE Infocom*, May 1996.
- [12] Z. Wang and J. Crowcroft. QoS routing for supporting multimedia applications. In *IEEE Journal of Selected Areas in Communications*, 14(7), 1996.