SpiderNet: An Integrated Peer-to-Peer Service Composition Framework *

Xiaohui Gu, Klara Nahrstedt, Bin Yu
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
Email:{xgu},{klara}, {binyu} @ cs.uiuc.edu

Abstract

Service composition is highly desirable in peer-to-peer (P2P) systems where application services are naturally dispersed on distributed peers. However, it is challenging to provide high quality and failure resilient service composition in P2P systems due to the decentralization requirement and dynamic peer arrivals/departures. In this paper, we present an integrated P2P service composition framework called SpiderNet to address the challenges. At service setup phase, SpiderNet performs a novel bounded composition probing protocol to provide scalable quality-aware and resource-efficient service composition in a fully distributed fashion. Moreover, SpiderNet supports directed acyclic graph composition topologies and explores exchangeable composition orders for enhanced service quality. During service runtime, SpiderNet provides proactive failure recovery to overcome dynamic changes (e.g., peer departures) in P2P systems. The proactive failure recovery scheme maintains a small number of dynamically selected backup compositions to achieve quick failure recovery for soft realtime streaming applications. We have implemented a prototype of SpiderNet and conducted extensive experiments using both large-scale simulations and wide-area network testbed. Experimental results show the feasibility and efficiency of the SpiderNet service composition solution for P2P systems.

1 Introduction

Peer-to-peer (P2P) systems are special Grid systems where Grid nodes called peers can communicate directly among themselves via application-level connections. Different from the conventional distributed systems, P2P systems are often fully decentralized and self-organizing. Recently, P2P systems have drawn much research attention with the popularity of various P2P file sharing systems such as Gnutella [1]. In this paper, we propose a *service-oriented* P2P system called *P2P service overlay* where peers can provide not only media files but also a number of application service components such as media transcoding and data filtering as well as application-level data routing, which is illustrated by Figure 1. The goal of such a P2P service overlay is to enable efficient *service*[1] sharing that is beyond the file sharing offered by current P2P systems [1]. Different from the conventional Grid systems [6, 14], each peer exports directly application service components for service sharing, which encompasses resource and data sharing. The advantage of such a service-oriented Grid system is to avoid both security problem in dynamic code uploading and expensive code/data migration across wide-area networks.

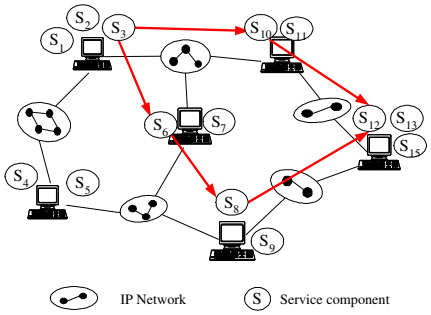

Figure 1. Peer-to-peer service overlay.

P2P service overlays are attractive since they promotes Internet-scale service sharing without any administration cost or centralized infrastructure support. New services can be flexibly composed from available service components

*This work was supported by the NASA grant under contract number NASA NAG 2-1406, NSF grant under contract number 9870736, 9970139, and EIA 99-72884EQ. Any opinions expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, NASA or U.S. Government.

[1]In this paper, service refers to application service that is a self-contained application unit.

based on the user's function and quality-of-service (QoS) requirements. Thus, P2P service overlays can achieve better scalability, manageability, and configurability in application service provisioning than the conventional client-server system model. Service composition across wide-area networks also becomes necessary in the P2P service overlay since service components are naturally distributed on different peers. Application examples of such P2P service overlays include: (1) *pervasive content distribution*, where the user can request adaptive content distribution to heterogeneous receivers with on-demand transformations and value-added customization; and (2) *collaborative scientific computation* [5], where geographically distributed research labs can leverage each other's solutions such as data analysis tools to conduct complex scientific experiments with lower development and computation cost.

Although previous research projects (e.g., [15, 19, 3, 18, 5]) have addressed the problems of service/resource selection and composition, they present the following major limitations when applied to P2P systems. First, most systems adopt a centralized approach that assumes the global system states information. However, the above assumption becomes impractical for a large-scale P2P system that often consists of thousands of peers dispersed across the wide-area network. Moreover, extended wide-area network delay and dynamic peer arrivals/departures can exacerbate the problem since it requires frequent information updates and thus large system overhead in order to alleviate states information imprecision. Second, most previous work does not consider the dynamic node changes that are common in P2P systems. Third, most existing solutions only support linear service composition with fixed composition order, which greatly limits the applicability and efficiency of service composition. We have presented the preliminary design of the P2P service composition framework in [10], which partially addressed the first two problems.

In this paper, we present an integrated P2P service composition framework called SpiderNet to address all of the above problems within a unified framework. For scalability, SpiderNet executes a novel *bounded composition probing* (BCP) protocol to provide fully decentralized QoS-aware service composition. In contrast to centralized schemes (e.g., [18, 12]), BCP performs on-demand selective states collection using a limited number of composition probes. The key observation behind our approach is that because of the service function constraints, QoS-aware service composition only needs the QoS and resource states information about those peers that can provide required service functions. These function-qualified peers often form a small sub-graph of the whole P2P network. Thus, it is more efficient to perform on-demand selective states collection than blindly maintain global states at each peer using expensive periodical states update. Compared to the conventional network probing, BCP has controllable overhead and considers service-specific requirements such as service function constraints and inter-service dependency/commutation relations.

Furthermore, SpiderNet provides efficient failure recovery to maintain the quality of composed services throughout the whole service session. SpiderNet adopts proactive failure recovery scheme to achieve *fast* failure recovery for soft realtime streaming applications. The proactive failure recovery approach maintains a *small* number of backup compositions for each active service session. The backup compositions are adaptively selected based on the conditions of the current composition and the user's QoS requirements. Thus, we can avoid the delay and overhead of triggering BCP to find a new composition if one of the maintained backup compositions can recover the failure.

We demonstrate the feasibility and efficiency of the SpiderNet service composition framework using prototype implementation. We conduct extensive experiments by evaluating the prototype on both large-scale simulation testbed and wide-area network testbed PlanetLab [2]. The experimental results show that SpiderNet can achieve near-optimal QoS-aware service composition performance with low overhead. Compared to the centralized approach that requires global states maintenance, SpiderNet can reduce the system overhead by more than one order of magnitude. Moreover, SpiderNet can achieve failure resilient service composition in a dynamic P2P network by maintaining a few number (e.g., less than 3) of backup compositions for each session.

The rest of the paper is organized as follows. Section 2 presents the system model. Section 3 briefly describes the decentralized service discovery scheme. Section 4 presents the initial decentralized QoS-aware service composition used by the service session setup phase. Section 5 describes the proactive failure recovery for maintaining the quality of composed services during service sessions. Section 6 presents the experimental results. Section 7 briefly discusses related work. Finally, the paper concludes in Section 8.

2 System Model

In this section, we first describe the SpiderNet system model. The SpiderNet system is implemented as a distributed middleware infrastructure deployed in wide-area networks, which can automatically map the user's composite service request into an instantiated distributed application service in the P2P service overlay.

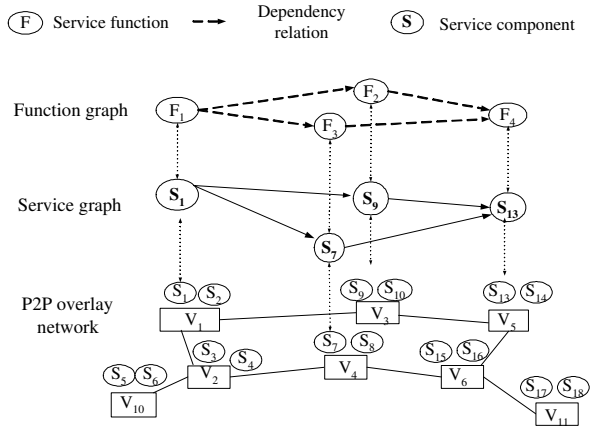

Figure 2. Service composition system architecture.

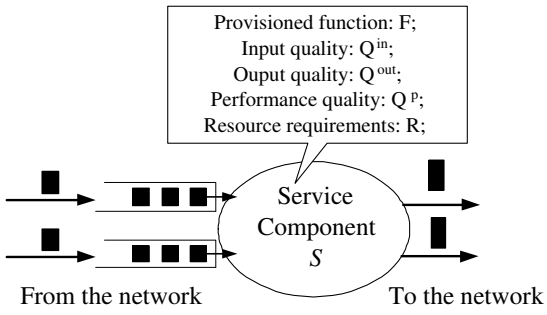

Figure 3. Service component model.

2.1 Composite Service Request

The composite service request consists of two parts: (1) function graph, shown by the top layer in Figure 2; and (2) quality-of-service (QoS) requirements $Q^{req} = [q_1^{req}, ..., q_m^{req}]$, where q_i is a quality parameter such as delay and data loss rate[2]. The function graph consists of required service functions $F_1, ..., F_k$ that are connected by dependency links and commutation links. The dependency link indicates that the output of one function is used as the input by its successor. The commutation link means that the composition order of two functions can be exchanged. For example, in Figure 4, function F_3 (e.g., color filter) can be exchanged with F_4 (e.g., image scaling). The user can specify the function graph using the visual specification environment such as QoSTalk [13, 23].

2.2 Distributed Application Service

The service component (s_i) is a self-contained application unit providing certain functionality, illustrated by Figure 3. Each service component includes one or more input queues for buffering input application data unit (ADU) from the network. Whenever the queue is not empty, the service component takes an input ADU from each input queue, processes them (e.g., audio mixing), and then sends the output ADU(s) to the network. Each service component accepts inputs with a quality level Q^{in} and generates outputs with a quality level Q^{out}, both of which are vectors of application-level quality parameters such as resolution and data format.

[2]For simplicity, we assume that all QoS metrics are additive since a multiplicative metric (e.g., loss rate) can be transformed into additive parameters using logarithmic function. Note that bandwidth is considered as resource metric that will be described later.

In order to process inputs and generate outputs, a specific amount of resources R is required, which is a vector of required resources (e.g., cpu, memory). Moreover, each service component is associated with a performance quality Q^p which is the same vector of performance parameters (e.g., delay) as the user's QoS requirements Q^{req}.

Service components can be composed into a service graph (λ) illustrated by the middle tier in Figure 2, which collectively deliver advanced composite services to the end-user. The link in the service graph is called service link that can be mapped to an overlay network path consisting of a set of overlay links. A service graph can be decomposed into multiple *branch paths*. For example, in Figure 2, the service graph consists of two branch paths $s_1 \rightarrow s_9 \rightarrow s_{13}$ and $s_1 \rightarrow s_7 \rightarrow s_{13}$.

2.3 Peer-to-Peer Service Overlay

We describe the P2P service overlay, illustrated by the bottom tier in Figure 2, using a directed graph $G = (V, E)$, where V represents the set of N peers $v_i, 1 \leq i \leq N$, and E represents the set of M overlay links $e_j, 1 \leq j \leq M$. Each peer provides a few number of service components. The overlay network topology can be either maintained as a topologically-aware overlay mesh [20] or dynamically constructed based on each peer's benefit [10]. However, our service composition system design is orthogonal to the underlying overlay topology.

2.4 Problem Description

We formulate the quality-aware service composition (QSC) problem in P2P service overlay as a two dimensional graph mapping problem, which is illustrated by Figure 4. In one dimension, we can derive different composition patterns from the original function graph by considering the commutation links. In the other dimension, we can map each service function into different functionally duplicated service components because of the inherent redundancy property of P2P systems [10]. These duplicated service

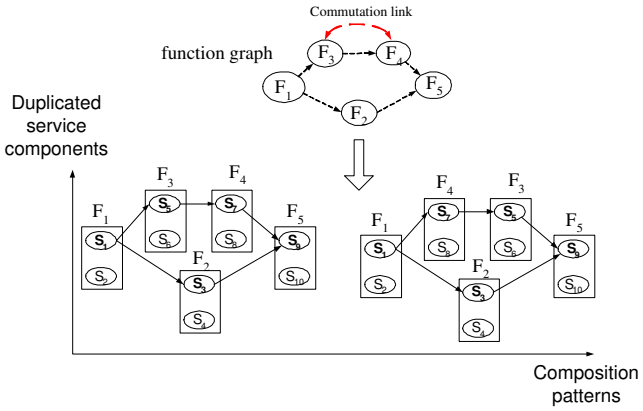

Figure 4. Two dimensional graph mapping problem.

components provide the same functionality but can have different QoS properties (e.g., service time) and available resources on the local peer host (e.g., CPU, memory). For example, in Figure 4, function F_1 can be mapped to two duplicated service components s_1 and s_2. Thus, we can derive different service graphs from the function graph by considering the above two dimensions. The QSC problem is to find the best mapping from the function graph to the best qualified service graph that satisfies the user's multi-constrained QoS requirements Q^{req} and achieves best load balancing in the current P2P service overlay. However, the QSC problem is NP-hard since it subsumes the multi-constrained path finding problem that has been proven to be NP-hard [8]. Thus, our goal is to provide efficient, fully decentralized service composition solution that is suitable for P2P computing environment.

3 Decentralized Service Discovery

This section briefly describes our decentralized service discovery substrate that allows each peer to locate services in the P2P service overlay without assuming a centralized service directory. We implement the decentralized service discovery based on the Pastry distributed hash table (DHT) system [22]. The basic function of the DHT system is to map a data key to a responsible peer. We realize the keyword-based service discovery by adding one meta-data layer on top of the DHT system. Due to the space limitation, we only briefly describe the decentralized service discovery as follows:

Service registration. When a peer wants to share a service component, it registers the service component by storing the component's static meta-data (e.g., location, input QoS, output QoS) into the DHT system. The peer first generates a key by applying a secure hash function on the function name of the service component. It then stores the meta-data into the DHT using the key. Because all functionally duplicated service component share the same function name and thus the key, the DHT system will store the meta-data list of the duplicated service components on the same DHT assigned peer.

Service discovery. When a peer wants to discover the list of service components matching certain function name, it can generate a key by applying the same secure hash function on the function name. Then, it can generate a query message using the key which will be routed to the assigned peer by the DHT system. The peer then returns the meta-data list of the duplicated service components to the requesting peer.

4 Initial Service Composition

In this section, we present the probing-based decentralized service composition solution used at the service session setup phase. We first introduce the bounded composition probing protocol. Then we describe the per-hop probe processing algorithm, followed by the optimal composition selection algorithm.

4.1 Bounded Composition Probing Protocol

Given a service composition request, the application sender[3] invokes the BCP protocol, which is illustrated by Figure 5. The BCP protocol includes four major steps:

Step 1. Initialize the probe. The source first generates a composition probing message, called probe, which is illustrated by Figure 5 (a). The probe carries the information of function graph and the user's QoS/resource requirements. To control the probing overhead, the probe carries a *probing budget* (β) that defines how many probes we could use for a composition request. The probing budget represents the trade-off between the probing overhead and composition optimality. Larger probing budget allows us to examine more candidate service graphs, which allows us to find a better qualified service graph. Thus, our solution can provide an adaptive composition solution with tunable performance by properly adjusting the probing budget. For example, we can use larger probing budget for the request with (1) higher priority, (2) stricter QoS constraints, or (3) more complex function. We can also adaptively adjust the probing budget based on the user feedbacks and historical information.

To achieve efficient composition probing, we associate a *probing quota* (α_i) with each function F_i in the function graph, which defines the number of duplicated service components to probe for F_i. The probing quota allows

[3]For simplicity, we use a unicast streaming application as an example.

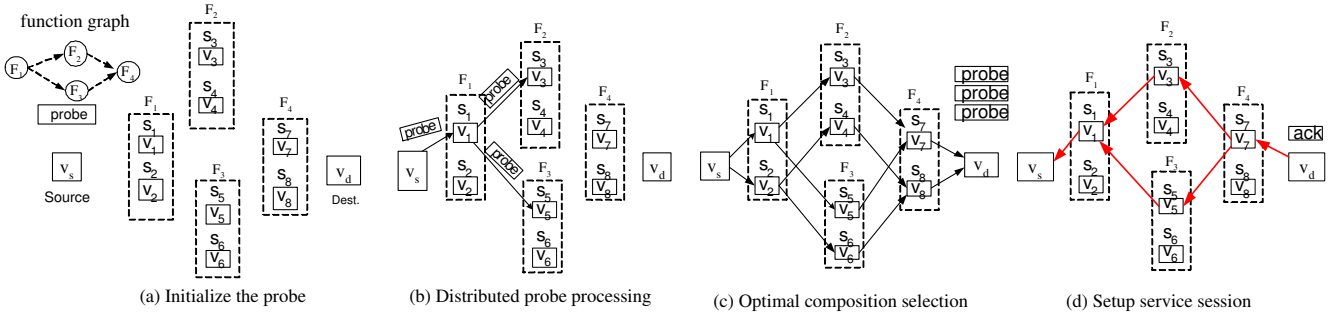

Figure 5. Bounded composition probing protocol.

us to achieve differentiated allocation of the probes among different functions. For example, we can assign higher probing quota for the function with more duplicated service components.

Step 2. Distributed probe processing. Each peer processes a probe independently using only local information until the probe arrives at the destination, illustrated by Figure 5 (b). The goal of hop-by-hop distributed probe processing is to collect needed information and perform intelligent parallel searching of multiple candidate service graphs. We will describe this step in detail in Section 4.2.

Step 3. Optimal composition selection. The destination collects the probes for a request with certain timeout period, illustrated by Figure 5 (c). It then selects the best qualified service graph based on the resource and QoS states collected by the probes. We will discuss this step in detail in Section 4.3.

Step 4. Setup service session. Finally, the destination sends an acknowledge message along the reversed selected service graph to confirm resource allocations and initialize service components at each intermediate peer, illustrated by Figure 5 (d). Then the application sender starts to stream application data units along the selected service graph. If no qualified service graph is found, the destination returns a failure message to the source directly.

4.2 Per-hop Probe Processing

We now describe the per-hop probe processing algorithm at a peer v_i which is illustrated by Figure 6. The per-hop probe processing mainly includes four steps:

Step 2.1 Resource/QoS check and soft resource allocation. When a peer receives a probe, it first check whether the QoS and resource values of the probed service graph already violate the user's requirements. If the accumulated QoS and resource values already violate the user's requirements, the probe is dropped immediately. Otherwise, the peer will temporarily allocate required resources to the expected application session. However, the resource allocation is *soft* since it will be cancelled after certain

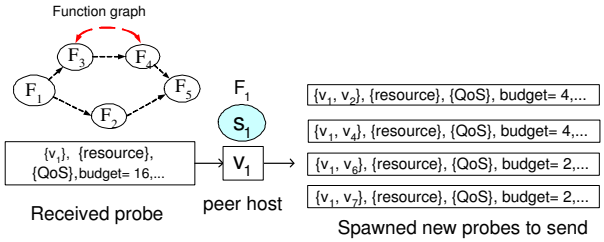

Figure 6. Per-hop probe processing operations.

timeout period if the peer does not receive a confirmation message. The purpose of this soft resource allocation is to avoid conflicted resource admission caused by concurrent probe processing. Thus, we can guarantee that the probed resources are still available at the end of the probing process.

Step 2.2 Derive next-hop functions. The peer derives next-hop functions according to the dependency and commutation relations in the function graph. All the functions dependent on the current function are considered as next-hop functions. For example, in Figure 6, the current function F_1 has two dependent next-hop functions F_2 and F_3. For each next-hop function F_k derived above, if there is a exchange link between F_k and F_l, then F_l is also considered as a possible next-hop function. For example, in Figure 6, since F_3 can be commutated with F_4, F_4 is also considered as the next-hop of F_1. The probing budget is proportionally distributed among next-hop functions according to their probing quotas.

Step 2.3 Select next-hop service components. For each next-hop function F_k, v_i first retrieves the meta-data of duplicated service components using the decentralized service discovery substrate described in Section 3. Let β_k denote the current probing budget for F_k. Let α_k define the probing quota allocated for F_k. Then the number of probes that can be used by F_k is denoted by $I_k = min(\beta_k, \alpha_k)$. Let Z_k denote the number of duplicated service components

for F_k. If $I_k \geq Z_k$, then v_i has enough probing budget to probe all the duplicated service components. Each probe has a new probing budget $\lfloor \frac{\beta_k}{Z_k} \rfloor$. However, if $I_k < Z_k$, we cannot probe all duplicated service components. Then v_i needs to select I_k most promising ones based on the local information. Currently, v_i uses a composite metric for next-hop service component selection, which comprehensively considers various local information such as network delay and available bandwidth to candidate next-hop service components, failure probability of candidate next-hop service components, and others. Finally, v_i spawns I_k new probes to examine the selected next-hop service components. Each new probe has a probing budget $\lfloor \frac{\beta_k}{I_k} \rfloor$.

Step 2.4 Set probe content. First, each new probe inherits the QoS and resource states from its parent probe. Then, v_i adds the local QoS states (e.g., Q^p of current-hop service component) and resource states (e.g., available CPU, memory on v_i) into the new probe, illustrated by Figure 6.

4.3 Optimal Composition Selection

The destination selects the best qualified service graph based on the information collected by the received probes. If the function graph has a linear path structure, each probe records a complete service composition. However, if the function graph has a DAG structure, each probe only collects the information for one composition branch. For example, in Figure 5, each probe traverses either branch $F_1 \rightarrow F_2 \rightarrow F_4$ or $F_1 \rightarrow F_3 \rightarrow F_4$. Thus, we need to first merge the branches into complete service graphs.

Next, the destination selects the qualified service graphs by comparing the QoS states of candidate service graphs with the user's QoS requirements. Then, the destination selects the best service graph from all the qualified ones based on the load balancing goal. For this purpose, we define a cost aggregation function ψ^λ as follows,

$$\psi^\lambda = \sum_{s_j/v_j \in \lambda} \sum_{i=1}^{n} w_i \cdot \frac{r_i^{s_j}}{ra_i^{v_j}} + w_{n+1} \cdot \sum_{\ell_j/\wp_j \in \lambda} \frac{b^{\ell_j}}{ba^{\wp_j}} \quad (1)$$

$$\sum_{i=1}^{n+1} w_i = 1, 0 \leq w_i \leq 1, 1 \leq i \leq n+1$$

where $r_i^{s_j}$ defines the requirement of s_j for the $i'th$ end-system resource type (e.g., CPU, memory), $ra_i^{v_j}$ defines the current resource availability on the peer v_j for the $i'th$ resource type, b^{ℓ_j} defines the bandwidth requirement on the service link ℓ_j, $ba^{\wp_j}$ defines the bandwidth availability on the underlying overlay network path $\wp_j$, and w_i represents the importance of different resource types. We can customize ψ^λ by assigning higher weights to more critical resource types. The rationale of using ψ^λ to evaluate the load balancing property of λ is that smaller ψ^λ means that the available resources along the service graph exceed the required resources by a larger margin. Thus, the service graph with the minimum ψ^λ can achieve the best load balancing in the current P2P computing environment.

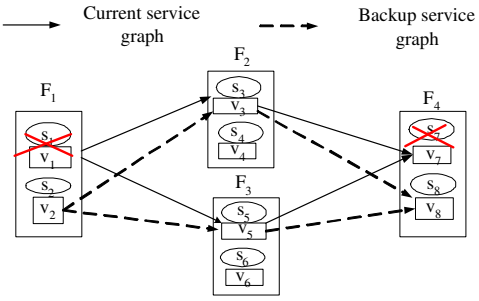

Figure 7. Proactive failure recovery for failure resilient service composition.

Finally, the source receives an acknowledge message carrying the information of the best service graph and a set of other qualified service graphs that can be used as backup service graphs for failure recovery, which will be introduced in the next section.

5 Proactive Failure Recovery

SpiderNet adopts a proactive approach to maintaining the quality of composed services during service runtime. The source maintains a small number of backup service graphs for each active service session. Thus, the source can quickly recover failures by switching from the broken service graph to one of the backup service graphs[4]. For example, in Figure 7, the source maintains one backup service graph for the service session. When the current service graph (shown in solid line) fails, the source can quickly switch to the backup service graph, shown in dash line, to recover the failure[5]. Different from the conventional multi-path routing, SpiderNet does not send real application data along multiple service graphs to achieve fault tolerance. Instead, the source only periodically sends low-rate measurement probes along these backup service graphs to monitor their liveness and QoS/resource conditions. The low-rate probing data is defined as the service graph maintenance overhead. The reactive failure recovery is triggered only when all backup service graphs become unqualified as well, which will invoke the BCP protocol to find a new qualified service graph.

[4]Due to the space limitation, we omit the discussion about the failure detection design details.

[5]We assume that the service component is either stateless or has only soft states that can be recovered quickly by software.

The advantages of the proactive failure recovery are two fold. First, we can achieve fast failure recovery by avoiding the delay of finding a new service graph if the backup service graph can be used, which is especially important for soft real time applications such as multimedia streaming. Second, we can reduce failure recovery overhead by avoiding invoking the relatively expensive BCP protocol to find a new service graph. Because P2P systems are highly dynamic, service graphs are prone to failures, especially for long-lived applications such as multimedia streaming. Thus, it is worthwhile to pay a small maintenance overhead for both fast failure recovery and greatly reduced composition probing overhead.

To achieve efficient proactive failure recovery, we need to answer two key questions: (1) how many backup service graphs should be maintained for a specific service session; and (2) which qualified service graphs should be selected as backup service graphs, which are described in the following sections.

5.1 Backup Service Graph Number

If we maintain too many backup service graphs, the maintenance overhead will be large. If we maintain too few backup service graphs, we may fail in providing the required quality assurances and failure resilience. Thus, the number of backup service graphs represents the tradeoff between the maintenance overhead and the quality of service composition. For efficiency, SpiderNet adopts an *adaptive* approach to deciding the number of backup service graphs based on the relationship between the QoS Q^λ and failure probability[6] F^λ of the current service graph λ and the user required QoS Q^{req} and failure probability F^{req}. Intuitively, if the QoS and failure probability of the current service graph are much better than the user's requirements, we could just maintain a few backup service graphs. Otherwise, we need to maintain more backup service graphs to achieve required QoS assurances and failure resilience. Formally, we can calculate the number of backup service graphs, denoted by γ, as follows,

$$\gamma = min(\lfloor U \cdot (\sum_{i=1}^{m} \frac{q_i^\lambda}{q_i^{req}} + \frac{F^\lambda}{F^{req}}) \rfloor, (C-1)) \qquad (2)$$

where U is a configurable system parameter defining the upper bound of backup service graph number, and C represents the total number of qualified service graphs found by initial service composition using BCP.

[6]Due to the space limitation, we omit the detailed discussion about the failure probability estimation. We can estimate the failure probability of a service graph using combinatorial approach if we assume peers' failure probabilities are independent or using performance modelling tools such as stochastic activity networks [4].

5.2 Backup Service Graph Selection

Given the number of backup service graphs, we need to decide which qualified service graphs should be selected as backup service graphs. On one hand, we want to achieve *failure resilience*, which implies that backup service graphs should be disjoint for failure independence; On the other hand, we want to achieve real time failure recovery, which implies that the backup service graph should be overlapped with the current service graph for fast failure recovery. Hence, we should carefully consider the tradeoff between the above two conflicting requirements. SpiderNet selects backup service graphs as follows:

- For each service component s_i on the current service graph λ, we select a qualified service graph that does not include s_i but has the largest overlap (i.e., largest number of common service components) with λ. Hence, if s_i fails, we can quickly recover λ by switching to the above backup service graph with lowest overhead.
- In order to handle multiple concurrent service component failures, we continue to select backup service graphs, which do not include every two service components, every three service components, and so forth.
- Under the constraint of backup service graph number, we may not be able to include all of the above desired backup service graphs. Thus, we start from selecting backup service graphs for recovering bottleneck service components, which have the largest failure probabilities.

6 Performance Evaluation

In this section, we evaluate the performance of SpiderNet using both large-scale simulations and prototype implementation evaluated in the wide-area network testbed, called PlanetLab [2].

6.1 Simulation Study

We have implemented an event-driven P2P service overlay simulator using C++. The simulator first uses the degree-based Internet topology generator Inet-3.0 [24] to generate a power-law graph to represent the IP-layer network. In our experiments, we used a 10,000 node IP network. We then randomly select 1000 nodes as SpiderNet nodes, which can be connected into different overlay topologies (e.g., mesh, power-law graph). Each node provides [1,3] service components whose provisioned tasks are selected from 200 pre-defined functions. The simulator performs IP-layer and overlay-layer data routing

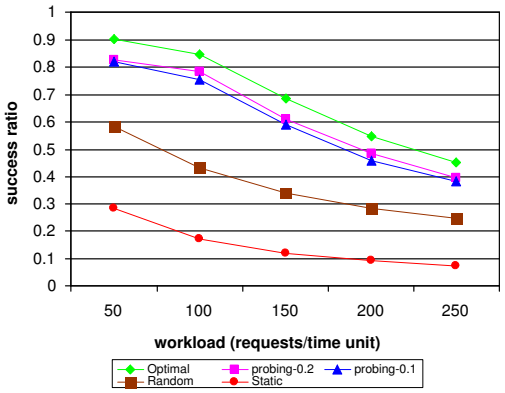

Figure 8. Performance comparison among different approaches.

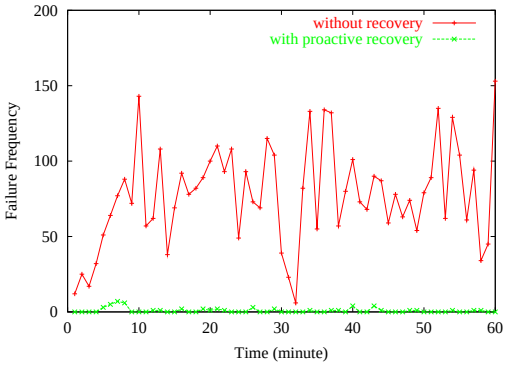

Figure 9. Failure frequency comparison in a dynamic P2P network.

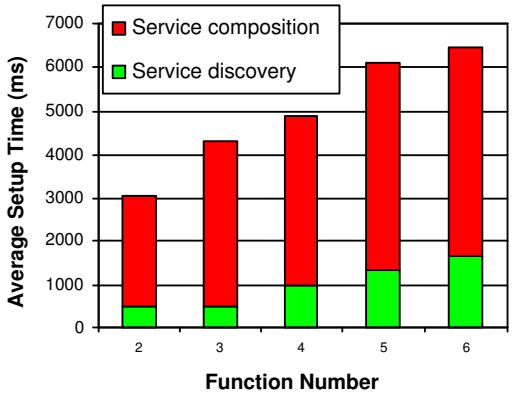

Figure 10. Service session setup time in wide-area networks.

using shortest path routing algorithm. During each time unit, certain number of composition requests are randomly generated on different peers. We define the metric "QoS success rate" to evaluate the performance of SpiderNet. A QoS-aware service composition is said to be successful, if and only if the composed service graph (1) satisfies the function graph requirements, (2) satisfies the user's resource requirements (e.g., CPU, network bandwidth) , and (3) satisfies the user's QoS requirements (e.g., delay, data loss rate). The composition success rate is calculated by $\frac{SuccessNumber}{RequestNumber}$.

For comparison, we also implement three other common approaches: *optimal*, *random*, and *static* algorithms. The optimal algorithm uses unbounded network flooding, which exhaustively searches all candidate service graphs to find the best qualified service graph. The random algorithm randomly selects a functionally qualified service component for each function node in the function graph. The static algorithm selects pre-defined service component for each function node in the function graph. Both random and static algorithms does not consider the user's QoS and resource requirements.

First, we compare the performance of different algorithms. Figure 8 illustrates the composition success rate achieved by different algorithms under different workload conditions. We used two variations of our scheme, "probing-0.2" and "probing-01", which uses 20% and 10% of the probes required by the optimal algorithm, respectively. Each round of simulation lasts 2000 time units. Each success rate is averaged over all the requests generated during 2000 time units simulation duration. We observe that our solution can achieve near-optimal performance with much lower overhead, and much better performance than random and static algorithms. Compared to the global-view-based centralized scheme, SpiderNet can achieve similar performance but with more than one order of magnitude less overhead since SpiderNet does not perform periodical global view maintenance.

Second, we evaluate the efficiency of our proactive failure recovery scheme. We use the metric "failure frequency" to define the number of failures occurred during each time unit. Figure 9 illustrates the failure frequency in a dynamic P2P network where 1% of peers randomly fail during each time unit. We observe that by maintaining on average 2.74 backup service graphs per session, the proactive failure recovery can recovery almost all the failures.

6.2 Prototype Implementation and Evaluation

We have implemented a prototype of the SpiderNet system. Each SpiderNet node software is a multi-threaded running system written in about 13K lines of java code.

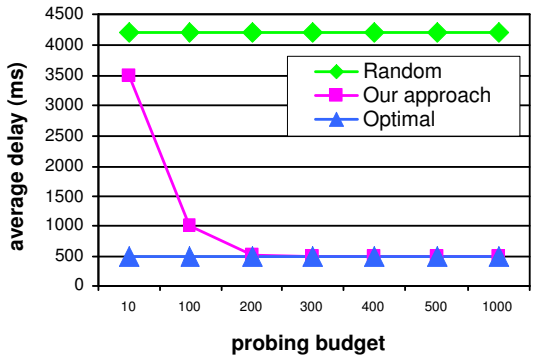

Figure 11. Performance comparison among random, SpiderNet, and optimal algorithms.

As proof-of-concept, we also implemented a set of multimedia service components to populate our P2P service overlay. Each service component provide one of the following six functions: (1) embedding weather forecast ticker; (2) embedding stock ticker; (3) up-scaling video frames; (4) down-scaling video frames; (5) extracting sub-image; and (6) re-quantification of video frames. We deploy one service component on each SpiderNet node, which is randomly selected from the above six multimedia service components. Our experiments use 102 Planetlab hosts that are distributed across U.S. and Europe. Thus, the average replication degree of each multimedia service is $102/6 = 17$. We then implement a customizable video streaming application on top of the SpiderNet service composition system. The customizable video streaming application allows the user to perform wide-area P2P video streaming with desired transformations and enriched content. We have deployed and evaluated the SpiderNet system with the video streaming application on the wide-area network testbed PlanetLab [2]. The end-application on each node periodically submits random service composition requests to the SpiderNet system.

First, we measure the service session setup time in the wide-area network, which includes (1) decentralized service discovery time, (2) initial service graph finding time using the bounded composition probing protocol, and (3) service session initialization time. Figure 10 illustrates the average service session setup time using more than 500 requests generated from 102 different PlanetLab hosts. The current prototype of the SpiderNet system can setup a service session within several seconds, which is acceptable for long-lived streaming applications that usually lasts tens of minutes or several hours. The above service setup time can be reduced with implementation implements and tunings.

Second, we compare the QoS provisioning performance of SpiderNet with the random and optimal algorithms. We consider service composition requiring three different functions. We ask different approaches to find the best qualified service composition that has minimum end-to-end service delay. Because each service function has on average 17 instances, the average number of probes required by the optimal algorithm is $17^3 = 4913$. As shown by Figure 11, the average service delay of the service graphs discovered by the SpiderNet reduces with a growing probing budget. When the probing budget is very low, SpiderNet degenerates into the random algorithm, so the overhead is low, but the service quality is not satisfactory. When larger probing budget is allowed, the service graph quality improves, and when the probing budget reaches a certain threshold, it asymptotically approaches the optimal performance. However, SpiderNet can achieve near optimal performance with much lower overhead (i.e., $200/4913 = 4\%$) than the unbounded flooding scheme performing exhaustive searching.

7 Related Work

In this section, we briefly compare related work with SpiderNet. Most P2P research projects have been focused on providing scalable data lookup solutions (e.g., [22, 21]) for efficient data sharing. In contrast, SpiderNet focuses on providing an integrated P2P service composition system to enable efficient service sharing. Different from data sharing, service sharing must consider additional application-specified service constraints such as function constraints and inter-service dependency/commutation relations.

Recently, several research projects (e.g., Ninja [9], SAHARA [17], CANS [7], Media Object Path [16], GriPhyN [5]) have addressed the problems of dynamic service composition under different context. In [11] and [12], we proposed two centralized service composition solutions for smart rooms and enterprise service overlay networks, respectively. SpiderNet differs from the above work by providing fully decentralized efficient service composition solution that is suitable for P2P systems. Moreover, SpiderNet focuses on addressing the challenge of scalable QoS and resource management issues, which is important for composing QoS sensitive distributed applications in P2P computing environments.

8 Conclusion and Future Work

We have presented an integrated P2P service composition framework called SpiderNet. The major contributions of this paper are summarized as follows. First, Spider-

Net provides fully decentralized QoS-aware and resource-efficient service composition using bounded composition probing. Second, SpiderNet provides proactive failure recovery to achieve failure resilient service composition. Third, SpiderNet achieves flexible service composition by supporting directed acyclic graph composition topologies and considering exchangeable composition orders to enhance the composed service's quality. Finally, we demonstrate the feasibility and efficiency of the SpiderNet system using both large-scale simulations and prototype implementation. In the future, we will integrate decentralized trust management into the current service composition framework to support secure service composition. We also plan to extend the current solution to support more expressive service composition semantics such as conditional branch.

References

[1] Gnutella. *http://gnutella.wego.com/*.

[2] the PlanetLab. *http://www.planet-lab.org/*.

[3] P. Chandra, Y. Chu, A. Fisher, J. Gao, C. Kosak, T.S. Eugene Ng, P. Steenkiste, E. Takahashi, and H. Zhang. Darwin: Customizable Resource Management for Value-Added Network Services. *IEEE Network Magazine, no. 1, vol. 15*, January 2001.

[4] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster. The Mobius Framework and Its Implementation. *IEEE Transactions on Software Engineering vol. 28, no. 10, pp. 956-969*, October 2002.

[5] E. Deelman and et al. Mapping Abstract Complex Workflows onto Grid Environments. *Journal of Grid Computing, vol. 1, no. 1, pp. 25-39*, 2003.

[6] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications, 11(2)*, 1997.

[7] X. Fu, W. Shi, A. Akkerman, and V. Karamcheti. CANS: Composable, Adaptive Network Services Infrastructure . *Proc. of 3rd USENIX Symposium on Internet Technologies and Systems*, March 2001.

[8] M. R. Garey and D. S. Johnson. Computers and Intractability. *A Guide to the Theory of NP-Completeness, Freeman, San Francisco*, 1979.

[9] S. D. Gribble and et al. The Ninja Architecture for Robust Internet-Scale Systems and Services. *Journal of Computer Networks, Volume 35, Issue 4*, March 2001.

[10] X. Gu and K. Nahrstedt. A Scalable QoS-Aware Service Aggregation Model for Peer-to-Peer Computing Grids. *Proc. of 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland*, July 2002.

[11] X. Gu and K. Nahrstedt. Dynamic QoS-Aware Multimedia Service Configuration in Ubiquitous Computing Environments. *Proc. of IEEE 22nd International Conference on Distributed Computing Systems (ICDCS 2002), Vienna, Austria*, July 2002.

[12] X. Gu, K. Nahrstedt, R. N. Chang, and C. Ward. QoS-Assured Service Composition in Managed Service Overlay Networks. *Proc. of IEEE 23nd International Conference on Distributed Computing Systems (ICDCS 2003), Providence, RI*, May 2003.

[13] X. Gu, K. Nahrstedt, W. Yuan, D. Wichadakul, and D. Xu. An XML-based QoS Enabling Language for the Web. *Journal of Visual Language and Computing (JVLC), Special Issue on Multimedia Language for the Web, 13(1), pp. 61-95*, February 2002.

[14] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. *Proc. of the 8th International Conference of Distributed Computing Systems*, June 1988.

[15] C. Liu, L. Yang, I. Foster, and D. Angulo. Design and Evaluation of a Resource Selection Framework for Grid Applications. *Proc. of the Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland*, June 2002.

[16] A. Nakao, L. Peterson, and A. Bavier. Constructing End-to-End Paths for Playing Media Objects. *Computer Networks 38(3), 373–389*, 2002.

[17] B. Raman and et al. The SAHARA Model for Service Composition Across Multiple Providers. *International Conference on Pervasive Computing (Pervasive 2002), Zurich, Switzerland*, August 2002.

[18] B. Raman and R. H. Katz. Load Balancing and Stability Issues in Algorithms for Service Composition. *Proc. of IEEE INFOCOM 2003, San Francisco, CA*, April 2003.

[19] R. Raman, M. Livny, and M. Solomon. Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching. *Proc. of the Symposium on High Performance Distributed Computing (HPDC-12), Seattle, WA*, June 2003.

[20] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-Aware Overlay Construction and Server Selection. *Proc. of Infocom 2002,New York City, N.Y.*, June 2002.

[21] M. Ripeanu and I. Foster. A Decentralized, Adaptive Replica Location Mechanism. *Proc. of 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland*, July 2002.

[22] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *Proc. of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, November 2001.

[23] D. Wichadakul, X. Gu, and K. Nahrstedt. A Programming Framework for Quality-Aware Ubiquitous Multimedia Applications. *Proc. of ACM Multimedia 2002, Juan Les Pins, France*, December 2002.

[24] J. Winick and S. Jamin. Inet3.0: Internet Topology Generator. *Tech Report UM-CSE-TR-456-02 (http://irl.eecs.umich.edu/jamin/)*, 2002.